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.
v	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. Even though Recurrent Neural Networks (RNNs), Convolutional Neural Networks (CNNs), and Transformers have variants to capture temporal dependencies across multiple frames, image degradation (i.e., motion blur, camera defocus, large pose variation, and object occlusions) make this a challenging task. Although state-of-the-art methods have refined feature selection, proposal, and aggregation modules, designed to capture temporal dependencies, State Space Models (SSMs) have emerged as a promising alternative to RNNs and attention-based mechanisms. Despite this, utilizing SSMs to model temporal relationships in video object detection remains to be investigated. As such, we present to our knowledge the first case of utilizing an SSM to capture the temporal relationship between consecutive frames in video object detection. Our contributions are threefold. First, we benchmark different RNNs against High Order Polynomial Projection Operator (HiPPO) a novel SSM framework on a simple sequence task, where HiPPO is also integrated as a memory mechanism for an RNN. Secondly, we recreate YoloV4 a classical CNN still-image object detector and extend it for video object detection to be used as a baseline comparison. Lastly, we show that RNNs yield sparse bounding box prediction. However, these predictions can result in a higher performance over the same number of predictions compared to the baseline when the model's confidence threshold is reduced. While an SSM formulation does not yield sparse predictions, mild performance degradations when modelling temporal dependencies are visible. Therefore, within this context, we compare YoloV4 a CNN with no ability to capture long-term dependencies, and variants that utilize an RNN and SSM formulation to model temporal dependencies.

Chapter 1

Introduction

Given the precise nature of the human visual system, which can localize, 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 localization 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 led to the very first 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 as shown in Fig. 1.1, making this a challenging task [3]. As previous frames in a sequence provide a rich history of contextual information about successive frames, efforts to explicitly model such dependencies are a natural choice to mitigate object appearance deteriorations in a frame.

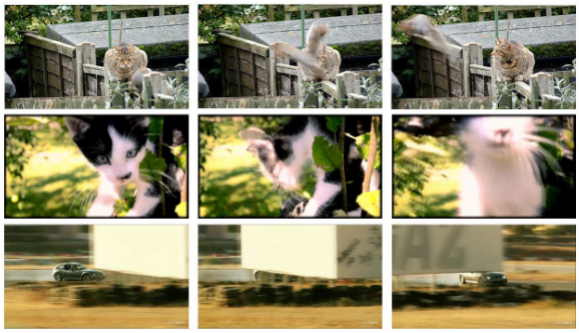


Figure 1.1: Challenges in video object detection. Motion blur, camera defocus and pose variation.

Early methods within the field focused on traditional computer vision techniques like background subtraction [4], motion detection [5] and handcrafted features [6; 7; 8; 9; 10; 11], which were limited in their ability to extract object features and model temporal dependencies. Over time, neural networks, particularly supervised deep learning models such as Convolutional Neural Networks (CNNs) [12], Recurrent Neural Networks (RNNs) [13] and Transformers [14; 15], have gradually taken the place of earlier approaches. Handcrafted features became less popular as CNNs automatically extracted a learned representation of complex features (i.e., patterns, textures, and structures) from images. Sequence models, such as RNNs [13; 16; 17] and Transformers [14] superseded early running-average and Mixtures of Gaussian Models (GMMs) [4] that maintained a dynamic model between consecutive frames as they were able to capture temporal dependencies over a longer context. Overall CNNs and sequence models overcame the limitations traditional techniques faced, which struggled to represent both spatial and temporal information effectively.

Given that CNNs effectively extract spatial features and sequence models (i.e., RNNs) handle temporal dependencies, we will focus on extending supervised deep learning for still-image detection to video object detection, addressing the challenge of identifying and localizing objects within sequences of frames over time. Unlike still-image detection, video object detection requires the model to account for temporal coherence and motion across consecutive frames. Features are extracted using a CNN and the temporal relationship is modelled using sequence models.

A simple example for still-image object detection is the detection of objects in natural images like photos. In general, the model is trained on an input x, that typically represents an image,

which may contain multiple objects that need to be classified and localized. Each image x can be represented as a multi-dimensional array of pixel values, depending on its resolution and color channels. For instance, an RGB image of size $280 \times 280 \times 3$ pixels is represented as a 3D array, where 280×280 represents the height and width of the image, and 3 corresponds to the three color channels (Red, Green, and Blue). The output y consists of a set of target variables that indicate both the class labels C of the objects and their corresponding bounding box coordinates. Each bounding box is usually defined by four coordinates: the x and y positions of the center, along with the width w and height h of the box. The categories C range from common objects like cars, people, and animals. Therefore, the goal in still-image object detection is to learn a mapping from the input image pixels x to the outputs or targets y that represent the 1) objects class labels and 2) their corresponding locations. In contrast, video object detection extends this task by incorporating temporal information to track and detect objects across consecutive frames, requiring the model to handle motion, occlusion, and other dynamic challenges. As such video object detection involves: 1) object classification, 2) bounding box regression and 3) modelling the temporal dependency of how objects move across consecutive frames.

Object Classification For instance, the classification aspect in object detection involves assigning a class category label C to each object in the image. Given an image, the goal is to identify what type of object is present within the image. For example, a model may be trained to classify objects into categories, where learned features are associated with a class label. The model learns to predict the correct class label by minimizing a classification loss, typically cross-entropy over a set of classes. The classification branch of the model outputs probabilities for each class, where the object is assigned the label corresponding to the highest-class probability.

Bounding Box Regression As for the regression component in still-image object detection, the model predicts the location and size of the object within an image. That is, given an image, the model learns to minimize the difference between the predicted bounding box coordinates based on learned features and the ground truth values, using a regression loss like the Mean Squared Error. Therefore, the regression component of the model outputs coordinates for the bounding box center x, y, including its width w and height h.

Temporal Dependencies However, to extend a still-image object detector that performs 1) object classification and 2) bounding box regression to videos, 3) the temporal dependencies between how objects move across different frames must be considered. As such, the role of the temporal module is to model the movement of objects or bounding boxes across consecutive frames. Unlike still-image detection, the model must account for changes in object position, scale, and appearance over time. By utilizing information from previous frames, the model can draw from its previous temporal context to track objects and learn how they unfold across time. Therefore, given a sequence of images, the model learns to minimize the difference between bounding box predictions for the current frame and the ground truth values, while also considering the predictions from previous frames. Sequence models, which are designed to handle sequential data by maintaining an internal state or memory that captures dependencies and patterns across different time steps, enabling them to learn from both past and present information in the sequence are an excellent choice for this task. Since they store information about previous time steps and how they unfold over time in some form of latent space, there is no explicit need to re-formalize loss. That is, since sequence models, inherently keep an internal memory or latent representation of previous context, that models the temporal dependencies across frames, the problem reduces to the same bounding box regression task as in the still-image scenario. Adjustment to the loss are not strictly necessary.

1.1 Object Detection

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.

1.1.1 Still-Image Object Detection

Region based frameworks 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 [18]. 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 must 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 interest. Among the most notable region-based proposal networks are R-CNN [19], Faster R-CNN [2] and Feature Pyramid Networks [20]. An example of a classical R-CNN can be seen in Fig. 1.2.

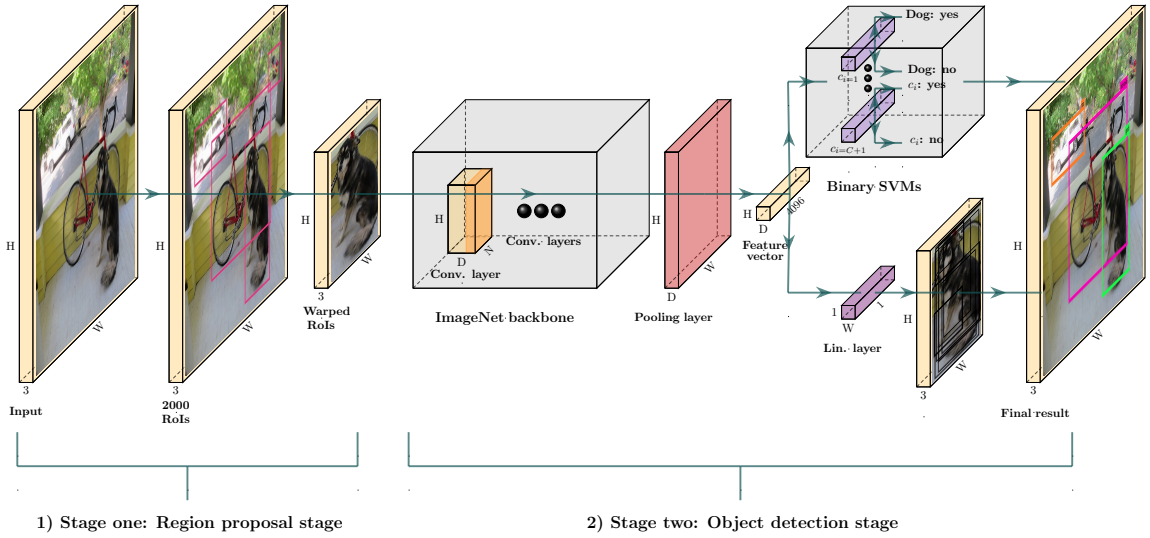


Figure 1.2: Generalized R-CNN still-image object detection workflow. The system is divided into a 1) a region proposal stage and 2) an object detection stage. The region proposal component generates regions of interest that serve as bounding box candidates, which are warped for further processing of 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 [21; 22; 23; 24], Single Shot MultiBox Detection (SSD) [25] and its variants [26; 27]. The generalized workflow of a typical regression-based model exemplified using YoloV1 can be seen in Fig. 1.3.

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 [28].

1.1.2 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 [29; 30; 31; 32; 33] and 2) object-wise [34; 35; 36] aggregation strategies. While pixel-level based methods aggregate information over entire

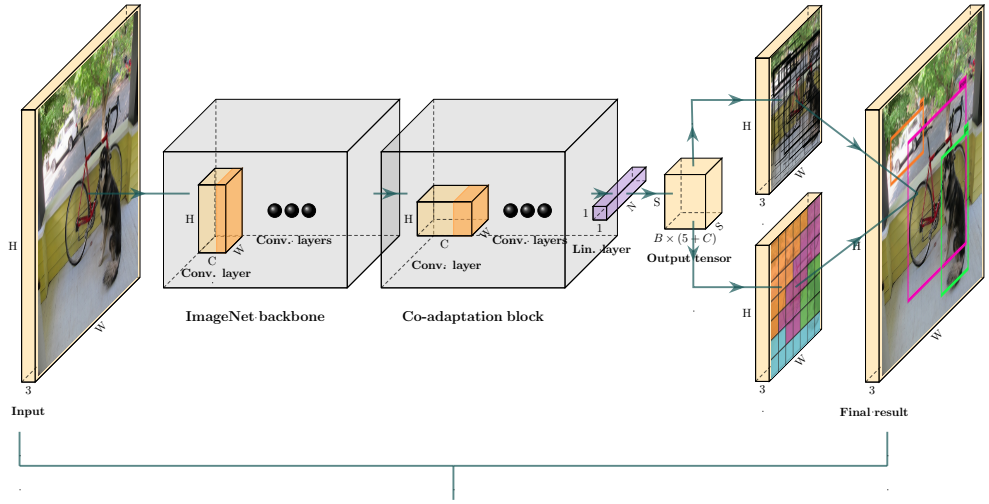


Figure 1.3: Generalized YoloV1 still-image object detection workflow. The system divides the image into an $S \times S$ grid. Then for each grid cell it predicts B bounding boxes, confidence for those boxes, and C class probabilities, where these predictions are encoded as an $S \times S(5 \times B + C)$ tensor.

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 [37; 38; 39; 40]. 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. Consequently, 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 [41; 29; 42; 43; 44] 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 [45; 46; 32] and attention-based models [33], 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, attention based mechanism on an object-level are also popular [47; 48; 49; 50; 51].

Recent state-of-the-art methods have explored several ways to improve video object detection. For example, YoloV [44] utilized a pre-trained still-image detector as a backbone and adds a specialized Feature Selection Module (FSM), followed by an attention-based Feature Aggregation Module (FAM) between backbone and the prediction head. In SLTnet [51] box features are first associated and aggregated by linking proposals from the same anchor box in nearby frames. Then, a specialized 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 [43] 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 these methods exploit feature selection or proposal module at a pixel or bounding box level to extract the temporal context, which are then aggregated. In contrast to these methods, our approach is much simpler

since we track object level features (i.e., bounding boxes) across time using an RNN and SSM.

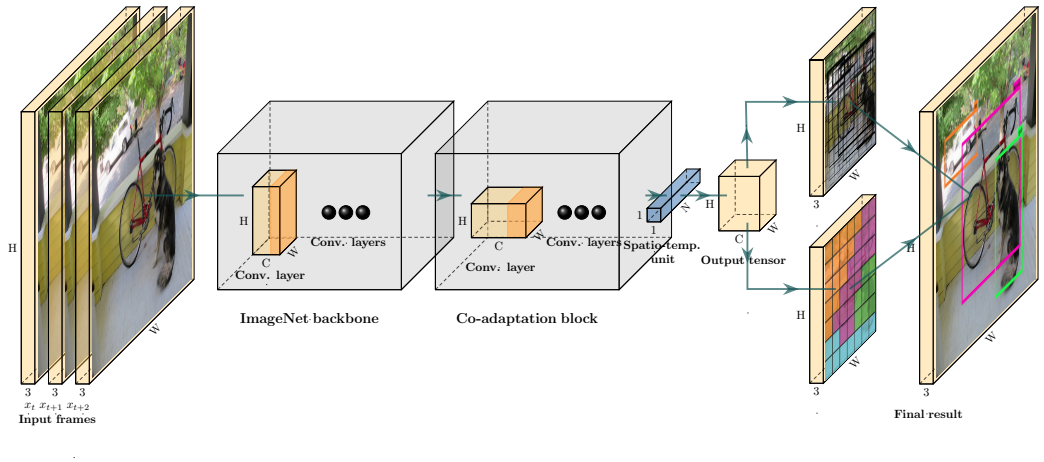


Figure 1.4: Adapted YoloV1 workflow for video object detection. The linear layer has been replaced by a sequence model i.e., an RNN represented as a blue square.

1.2 Roadmap

This document has a natural progression, where in Chapter 1 we started by introducing the problem of both still-image and video object detection. Popular approaches for both object detection on still-images and videos are covered to provide the reader with the necessary background knowledge. Sequence models are briefly explained to model the temporal relationship between successive frames in videos.

In Chapter 2 the Sequence Model framework is introduced in more detail. RNNs [13], including more advanced variants that utilize gating mechanisms (i.e, GRU[16], LSTM [17], UR-LSTM [52]) to address the exploding and vanishing gradient problem during optimization are reviewed in detail. Modes of failures during the optimization process are identified by analyzing Back-Propagation Through Time (BPTT) in depth. Continuous-time and discrete-time SSMs are covered, before presenting HiPPO [53] a novel SSM formulation as a solution to address the exploding and vanishing gradient problem.

Chapter 3: Method describes the approaches used for still-image and video object detection using YoloV4 [23]. We detail YoloV4 loss, architecture and the training procedure. Adaptations to YoloV4 are made to handle sequential data, using RNNs, namely UrYoloV4 (YoloV4 + UR-LSTM) and HoloV4 (YoloV4 + HippoLSTM), which are designed to capture temporal dependencies in video data.

In Chapter 4: Experiments various RNNs, HiPPO, and a HiPPO-LSTM are first evaluated on their performance with respect to the Sequential MNIST task, a simplified long-context challenge. The two winning candidates are used to model the temporal relation in video data. Subsequently, we validate our re-implementation of the YoloV4 framework for still-image detection on MS COCO [54] and extend it for video object detection on ImageNet VID [55]. For video object detection, the YoloV4 framework is used as a baseline comparison against YoloV4 adaptations that utilize a UR-LSTM and HiPPO-LSTM to model temporal dependencies. We call these extensions to the YoloV4 framework, UrYoloV4 and HoloV4. Evaluation results on the ImageNet VID [55] across all our video detectors be found in this chapter as well. We hope that this comparative analysis with the baseline model will highlight the improvements in accuracy and temporal coherence achieved by incorporating sequence models into the YOLO framework.

In Chapter 5 we discuss three sets of experiments: 1) a comparison of various sequence models on a simplified long-context task, 2) still-image detection, and 3) video object detection. We provide a detailed analysis of the results, highlighting their implications and explaining the models' behaviors in relation to their specific tasks. Additionally, we address some limitations observed in the video

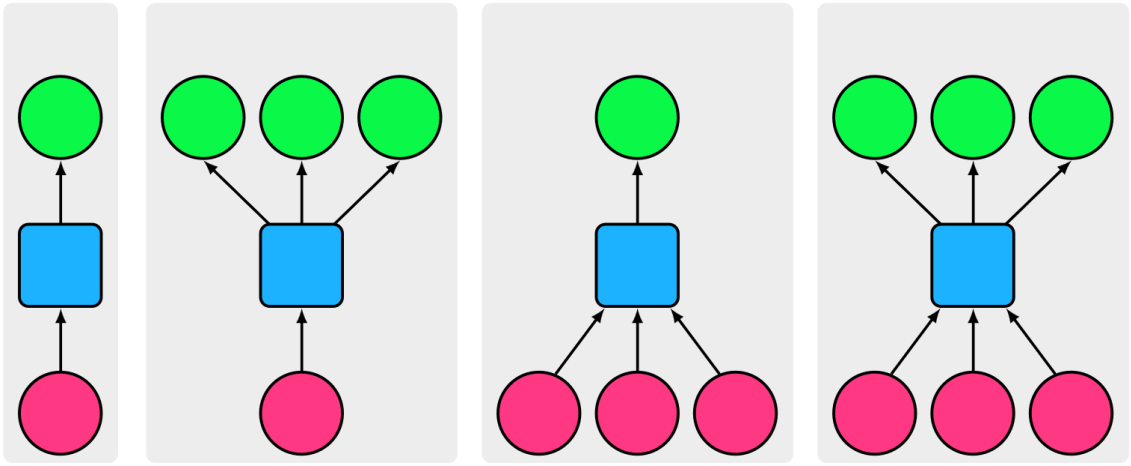
object detection experiments, offering insights into potential areas for improvement.

Finally, in Chapter 6, we present our conclusion, with a primary focus on the task of video object detection. We summarize the key findings, discuss the broader implications of our results, and outline directions for future work in this area.

Chapter 2

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. 2.1(a)). Similarly, a model that maps a single input to many outputs, forms a one-to-many relation and is called a "one-to-many" model (see Fig. 2.1(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 designed to handle multiple inputs represented as a sequence. Whether they map the input sequence to one output (see Fig. 2.1(c)) or multiple outputs (see Fig. 2.1(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 2.1: Simplified examples for different types of sequence models. Input vectors are highlighted in red, sequence models are blue and output vectors are in green.

2.1 Recurrent Neural Network

Recurrent neural networks (RNNs) [13; 17; 16; 52] are a class of neural networks that belong to the family of sequence models. They process data sequentially and keep 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 [56; 57] or gesture recognition [58; 59], 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. 2.2. 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. (2.1):

$$\begin{aligned} \mathbf{h}_t \in \mathbb{R}^d &= f_W\left(\mathbf{h}_{t-1}, \mathbf{x}_t\right) \\ &= \phi\left(\mathbf{x}_t W_{ih}^T + \mathbf{b}_{ih} + \mathbf{h}_{t-1} W_{hh}^T + \mathbf{b}_{hh}\right), \end{aligned} \tag{2.1}$$

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.

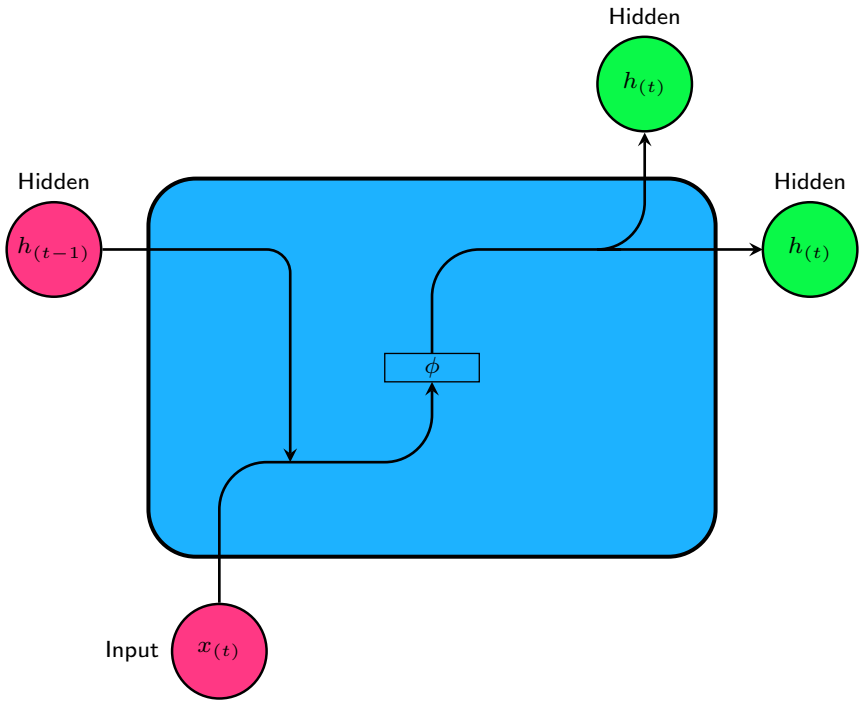


Figure 2.2: Gating mechanism of a simple RNN cell. The cell takes as input the input $\mathbf{x_t}$ at the current time step t and the hidden state at the previous time step $t-1$, returning the hidden state at the current time step $\mathbf{h_t}$. Inputs are colored red, outputs green, and the cell including the gating mechanism are in blue.

2.1.1 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: $(\hat{y}_t - y_t)$. The overall loss $\mathcal{L}(\hat{y}, y)$ is typically designed to be non-negative and usually based on squared errors, absolute differences, or other measures that ensure each time step's contribution to the loss remains positive. It is usually the sum of the time step specific loss found in the range of interest $[t, T]$ given by:

$$\mathcal{L}(\hat{\mathbf{y}}, \mathbf{y}) = \sum_{t=1}^{T} \mathcal{L}\left(\hat{\mathbf{y}}_t, \mathbf{y}_t\right) \tag{2.2}$$

Training of an unfolded recurrent neural network as illustrated in Fig. 2.3, is done across multiple time steps using a generalization of the backpropagation algorithm, 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 [60]. Given the total number of time steps T, the gradient of the loss with respect to the weights is then given by:

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial W} &= \sum_{t=1}^{T} \frac{\partial \mathcal{L}_t}{\partial W} \\ &= \sum_{t=1}^{T} \frac{\partial \mathcal{L}_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_1} \frac{\partial h_1}{\partial W} \end{aligned} \tag{2.3}$$

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 event at time step t back to one at a time step $k = 1$, as:

$$\begin{aligned} \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_1} &= \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}} \frac{\partial \mathbf{h}_{t-1}}{\partial \mathbf{h}_{t-2}} \cdots \frac{\partial \mathbf{h}_{1+1}}{\partial \mathbf{h}_1} \\ &= \prod_{i=k+1}^{t} \frac{\partial \mathbf{h}_i}{\partial \mathbf{h}_{i-1}} \end{aligned} \tag{2.4}$$

This product of subsequent linking Jacobians in Eq. (2.4), 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. (2.1), yields $\mathbf{W}\phi'\left(\mathbf{h}_{t-1}\right)$, where ϕ denotes an activation function.

As such:

$$\prod_{i=k+1}^{t} \frac{\partial \mathbf{h}_i}{\partial \mathbf{h}_{i-1}} = \prod_{i=k+1}^{t} \mathbf{W}\phi'\left(\mathbf{h}_{i-1}\right) \tag{2.5}$$

Performing an eigenvalue decomposition on the Jacobian $\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}}$ given by $\mathbf{W}\phi'\left(\mathbf{h}_{t-1}\right)$, the eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_n$ and corresponding eigenvectors $v_1, v_2, \ldots, v_n$ 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. (2.5) ensures 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 [61], 2) gradient regularization [61], 3) gating mechanism (i.e., Gate Recurrent Unit [16] or Long Short-Term Memory [17] cells) and 4) orthogonal initialization methods [62]. To exemplify gating mechanisms, the following sections will cover a Long Short-Term Memory cell and will also examine a method that utilizes orthogonality to stabilize gradients.

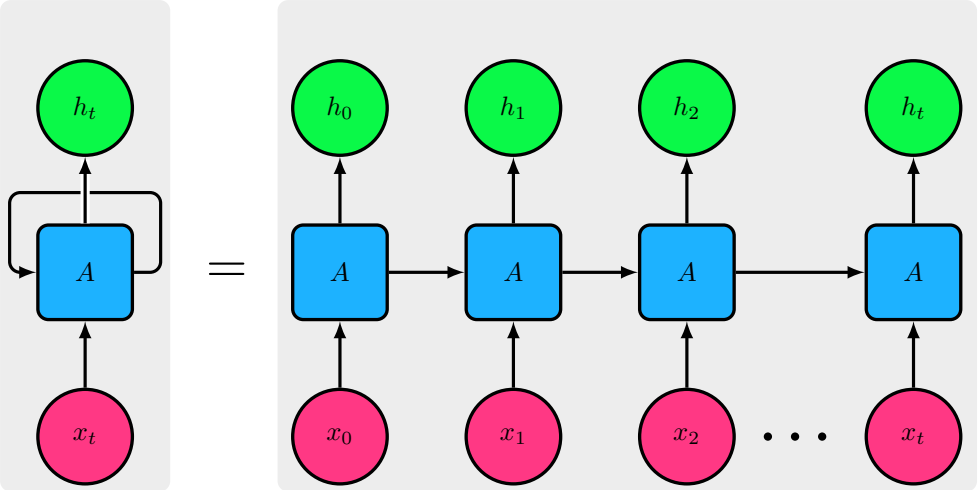


Figure 2.3: A recurrent neural network that takes input $\mathbf{x}_t$ and the hidden state from the previous time step h_{t-1} to produce the current hidden state $\mathbf{h}_t$ for every time step t (Left). The same network and its recursion can be unfolded through time (Right). Inputs are colored red, outputs green, and the RNN cell is blue.

2.1.2 Long Short-Term Memory

A Long Short-Term Memory (LSTM) [17] unit like the Gates Recurrent Unit (GRU) [16], makes use of sophisticated gating mechanisms that control the flow of information to and from the unit. By shutting the gates, these units can 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. 2.4. 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. (2.6)). 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 a 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. (2.7)). 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. (2.8)). 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. (2.9)). 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. (2.10)). 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. (2.6) - (2.11):

$$\begin{aligned}
\mathbf{i}_t &= f_W\left(x_t, h_{t-1}\right) && (2.6)\\
&= \sigma\left(W_{ii}x_t + b_{ii} + W_{hi}h_{t-1} + b_{hi}\right)\\
\mathbf{f}_t &= f_W\left(x_t, h_{t-1}\right) && (2.7)\\
&= \sigma\left(W_{if}x_t + b_{if} + W_{hf}h_{t-1} + b_{hf}\right)\\
\mathbf{g}_t &= f_W\left(x_t, h_{t-1}\right) && (2.8)\\
&= \tanh\left(W_{ig}x_t + b_{ig} + W_{hg}h_{t-1} + b_{hg}\right)\\
\mathbf{o}_t &= f_W\left(x_t, h_{t-1}\right) && (2.9)\\
&= \sigma\left(W_{io}x_t + b_{io} + W_{ho}h_{t-1} + b_{ho}\right)\\
\mathbf{c}_t \in \mathbb{R}^d &= f_t \odot c_{t-1} + i_t \odot g_t && (2.10)\\
\mathbf{h}_t \in \mathbb{R}^d &= o_t \odot \tanh\left(c_t\right), && (2.11)
\end{aligned}$$

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.

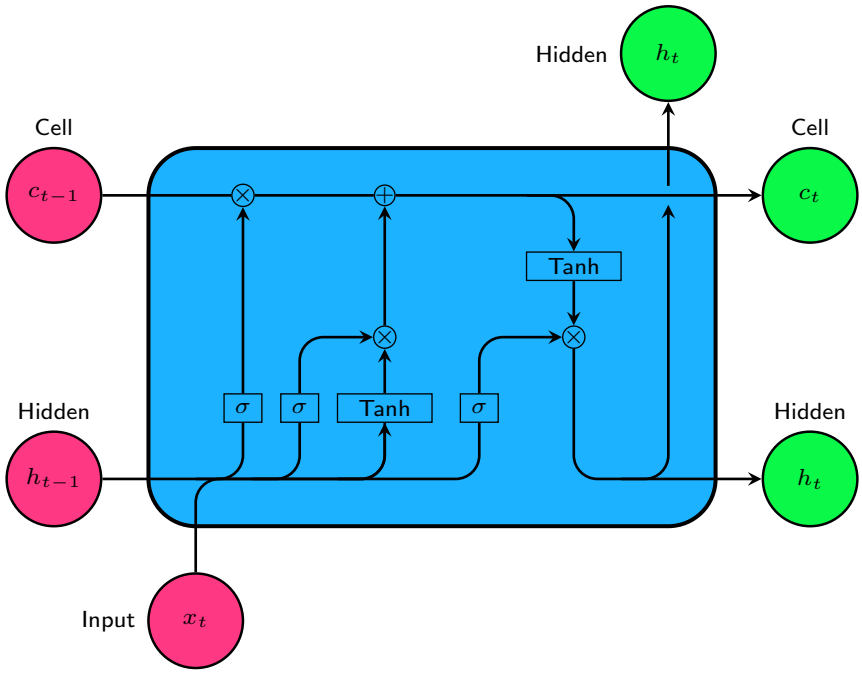


Figure 2.4: Gating mechanism of an LSTM cell. The cell takes as input the input at the current time step t, the hidden state at the previous time step $t-1$ and cell state of the previous time step $t-1$, returning the hidden state and cell state of the current time step.

2.1.3 Cell state: Backpropagation Through Time

As with the vanilla RNN, we can assess the gradient flow through the network over time to see whether it alleviates the problem of vanishing or exploding gradients. To do this, we examine the partial derivative of the LSTM cell state $\mathbf{c}_t$ at time step t w.r.t the first cell state $\mathbf{c}_1$, which also involves the product of Jacobians $\frac{\partial \mathbf{c}_t}{\partial \mathbf{c}_{t-1}}$ over multiple time steps, linking an event at time step t back to one at a time step $k = 1$, as:

$$\begin{aligned} \frac{\partial \mathbf{c}_t}{\partial \mathbf{c}_1} &= \frac{\partial \mathbf{c}_t}{\partial \mathbf{c}_{t-1}} \frac{\partial \mathbf{c}_{t-1}}{\partial \mathbf{c}_{t-2}} \cdots \frac{\partial \mathbf{c}_{1+1}}{\partial \mathbf{c}_1} \\ &= \prod_{i=k+1}^{t} \frac{\partial \mathbf{c}_i}{\partial \mathbf{c}_{i-1}} \end{aligned} \tag{2.12}$$

If we examine 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, we find:

$$\begin{aligned}\frac{\partial \mathbf{c}_t}{\partial \mathbf{c}_{t-1}} &= \frac{\partial \phi\left(\mathbf{u_t}, \mathbf{c}_{t-1}\right)}{\partial \mathbf{c}_{t-1}} \\ &= \frac{\partial \sigma\left(\mathbf{u}_t\right) \mathbf{c}_{t-1}}{\partial \mathbf{c}_{t-1}} \\ &= \mathbf{c}_{t-1} \underbrace{\frac{\partial \sigma\left(\mathbf{u}_{\mathrm{t}}\right)}{\partial \mathbf{c}_{\mathrm{t}-1}}}_{=1} + \underbrace{\frac{\partial s_{\mathrm{t}-1}}{\partial \mathbf{c}_{\mathrm{t}-1}}}_{=0} \sigma\left(\mathbf{u}_{\mathrm{t}}\right) \\ &= \sigma\left(\mathbf{u}_{\mathrm{t}}\right)\end{aligned} \tag{2.13}$$

Thus:

$$\frac{\partial \mathbf{c}_t}{\partial \mathbf{c}_{t-1}} = \prod_{t=k+1}^{t} \sigma\left(\mathbf{u}_{\mathrm{t}}\right) \tag{2.14}$$

Stability between consecutive updates in the cell state is introduced via the sigmoid activation, binding the term between 0 and 1, preventing the exploding gradient problem [63]. Through multiple products of values below 1, it can still approach zero over long sequences [52]. 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.

2.1.4 Uniform Refine Long Short-Term Memory

A Uniform Refine Long Short-Term Memory (UR-LSTM) [52] is a refinement of the LSTM gating mechanism in combination with a uniform initialization (U). Standard gate initialization regimes can prevent learning of long-term temporal dependencies [64] 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 decay exponentially w.r.t. to k time steps via $\mathbf{f}_t^k$ as shown in Eq. (2.14) and thus saturate gradients. Initializing the forget-gate can alleviate the problem of exponential decay, effectively increasing the timespan over which dependencies can be learned [65]. 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. (2.7)). 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. 2.5 that will be used instead of the original update of the LSTM (see Eq. (2.11)). As a result, the refine mechanism can be integrated to modulate the gating of an LSTM cell as shown in Fig. 2.6. Note that the 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 UR-LSTM adds two small modifications to the vanilla LSTM. First, it introduces an initialization strategy on the forget-gate f biases, according to a uniform distribution $\mathcal{U}(0, 1)$:

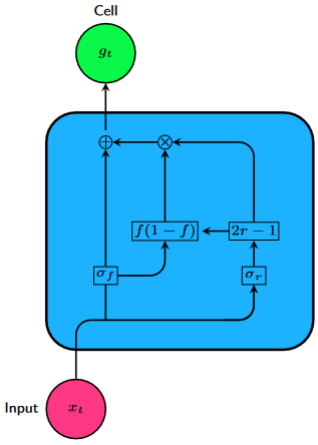


Figure 2.5: Refine gate used for the UR-LSTM. The mechanism takes as input x_t at time step t. Note that the σ_f denotes the outputs of the forget-gate f and σ_r the output of the refine-gate r.

$$b_f \sim \mathcal{U}(-\frac{1}{\sqrt{\mathrm{p}}}, \frac{1}{\sqrt{\mathrm{p}}}), \tag{2.15}$$

p denoting the hidden size.

Secondly, a refine gate r, which allows for better gradient flow by parameterizing the forget-gate f to not saturate as quickly is introduced. Formally, the full mechanism of the refine gate is defined

in Eq. (2.17) - (2.18). Let u be the get-gate and $\mathbf{u}_t$ its output values from the original LSTM pathway as defined in Eq. (2.8), then the UR-LSTM gating can be defined as:

$$\mathbf{r}_t = \sigma\left(f_{rW}\left(x_t, h_{t-1}\right)\right) \tag{2.16}$$

$$\mathbf{g}_t = r_t \odot \left(1 - \left(1 - f_t\right)^2\right) + \left(1 - r_t\right) \odot f_t^2 \tag{2.17}$$

$$\mathbf{c}_t \in \mathbb{R}^d = \mathbf{g}_t \mathbf{c}_{t-1} + \left(1 - \mathbf{g}_t\right) \mathbf{u}_t, \tag{2.18}$$

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.

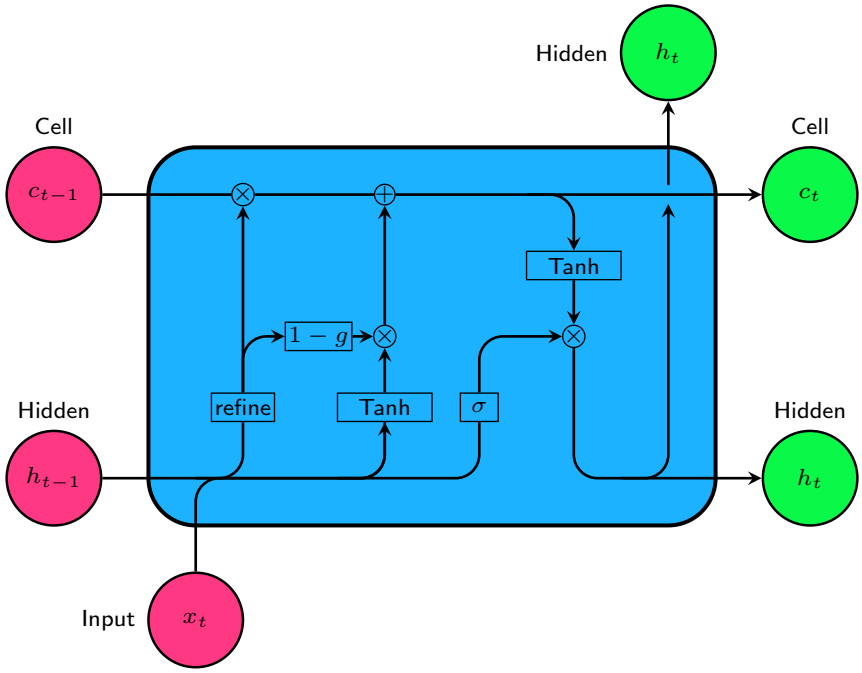


Figure 2.6: Gating mechanism of a UR-LSTM cell. The cell takes as input the input at the current time step t, the hidden state at the previous time step $t-1$ and cell state of the previous time step $t-1$, returning the hidden state and cell state of the current time step.

2.1.5 Refined Cell State: Backpropagation Through Time

To assess whether the UR-LSTM addresses the gradient problem, we examine the partial derivatives of the refined cell state, allowing us to understand how gradients flow over 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:

$$\begin{aligned}\frac{\partial \mathbf{c}_t}{\partial \mathbf{c}_1} &= \frac{\partial \mathbf{c}_t}{\partial \mathbf{c}_{t-1}} \frac{\partial \mathbf{c}_{t-1}}{\partial \mathbf{c}_{t-2}} \cdots \frac{\partial \mathbf{c}_{1+1}}{\partial \mathbf{c}_1} \\ &= \prod_{i=k+1}^{t} \frac{\partial \mathbf{c}_i}{\partial \mathbf{c}_{i-1}}\end{aligned} \tag{2.19}$$

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

$$\begin{aligned}\frac{\partial \mathbf{c}_t}{\partial \mathbf{c}_{t-1}} &= \frac{\partial \mathbf{g}_t \mathbf{c}_{t-1} + (1 - \mathbf{g}_t)\mathbf{u}_t}{\partial \mathbf{c}_{t-1}} \\ &= \frac{\partial (\mathbf{g}_t \mathbf{c}_{t-1})}{\partial \mathbf{c}_{t-1}} + \frac{\partial ((1 - \mathbf{g}_t)\mathbf{u}_t)}{\partial \mathbf{c}_{t-1}} \\ &= \mathbf{g}_t \underbrace{\frac{\partial \mathbf{c}_{t-1}}{\partial \mathbf{c}_{t-1}}}_{=1} + (1 - \mathbf{g}_t) \underbrace{\frac{\partial \mathbf{u}_t}{\partial \mathbf{c}_{t-1}}}_{=0} \\ &= \mathbf{g}_t \cdot 1 + (1 - \mathbf{g}_t) \cdot 0 \\ &= \mathbf{g}_t + 0 \\ &= \mathbf{g}_t\end{aligned} \tag{2.20}$$

Thus:

$$\frac{\partial \mathbf{c}_t}{\partial \mathbf{c}_{t-1}} = \prod_{t=k+1}^{t} \mathbf{g}_t \tag{2.21}$$

Like 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, due to eigenvalue $\lambda_1 < 1$, the gradients will still vanish once g_t is below a certain threshold.

2.2 State Space Model

Recently state space models (SSMs) [53; 66; 67] 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 [68]. An attractive feature of SSMs is that compared to transformers [14], which have a quadratic memory and computational cost with respect to the input length both at training and inference, 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 [53], 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.

2.2.1 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. (2.22). 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. (2.23).

$$x'(t) = A(t)\mathbf{x}(t) + B(t)\mathbf{u}(t) \tag{2.22}$$

$$\mathbf{y}(t) = C(t)\mathbf{x}(t) + D(t)\mathbf{u}(t), \tag{2.23}$$

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. 2.7 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) [69; 70]. 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.

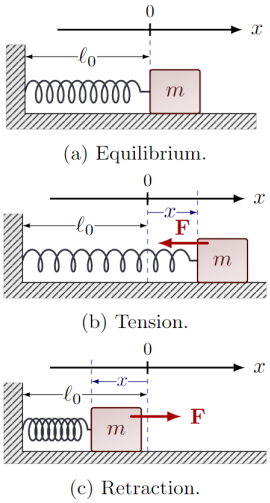


Figure 2.7: Representation of various dynamic states of a spring-mass system.

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 [71]. 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 Newton's Second Law:

$$F = ma, \tag{2.24}$$

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:

$$F = -kx, \tag{2.25}$$

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:

$$ma = -kx(t) \tag{2.26}$$
$$mx''(t) = -kx(t) \tag{2.27}$$
$$x''(t) = -\frac{k}{m}x(t), \tag{2.28}$$

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:

$$\begin{bmatrix} x'(t) \\ v'(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & 0 \end{bmatrix} \begin{bmatrix} x(t) \\ v(t) \end{bmatrix}, \tag{2.29}$$

where A(t), the system matrix is:

$$A(t) = \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & 0 \end{bmatrix} \tag{2.30}$$

Adding some external force (or input signal) $\mathbf{u}(t)$ that acts on the system dynamics and is regulated by the input matrix $B(t)$, we revisit Eq. 2.26 - 2.28:

$$ma = -kx(t) + u(t) \tag{2.31}$$
$$mx''(t) = -kx(t) = u(t) \tag{2.32}$$
$$x''(t) = -\frac{k}{m}x(t) + \frac{u(t)}{m} \tag{2.33}$$

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:

$$\begin{bmatrix} x'(t) \\ v'(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & 0 \end{bmatrix} \begin{bmatrix} x(t) \\ v(t) \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix} u(t), \tag{2.34}$$

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

$$B(t) = \begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix} u(t), \tag{2.35}$$

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:

$$C(t) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{2.36}$$

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:

$$D(t) = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{2.37}$$

2.2.2 Discrete-Time State Space Model

To represent a discrete-time SSM as a time dynamic system, the continuous input signal u_t needs to be discretized using a step size Δt, which represents the resolution of the input signal. Conceptually, the step size can be seen as the resolution of the input signal, allowing us to represent the discretized input u_k as samples from the underlying continuous signal u_t, where $u_k = u(k\Delta t)$.

The discretization step is often accomplished through Forward Euler [72], Zero-Order Hold (ZOH) [73], or the Bilinear Transform (also known as the Tustin Transform) [74], which approximates the continuous state matrix A and the continuous state input matrix B into a discrete-time format. As Forward Euler is the simplest and assumes constant system dynamics over time, we will use it for illustrating the conversion from continuous form to discrete form. Using the Forward Euler method, the continuous-time system:

$$x'(t) = A(t)\mathbf{x}(t) + B(t)\mathbf{u}(t) \tag{2.38}$$

is approximated as:

$$x_{k+1} = (I + \Delta tA)x_k + \Delta tBu_k \tag{2.39}$$

with Δt representing the discretization rate or step size of discretization of the continuous signal, A_k and B_k being the system and input matrix at time step k respectively.

As such, 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 [69; 70]. They map a 1-D input signal $\mathbf{u}_k$ to an N-D latent space $\mathbf{x}_k$, which can then also be projected back to a 1-D output signal $\mathbf{y}_k$ as shown in Eq. (2.40) and Eq. (2.41).

$$\mathbf{x}_{k+1} = A_k\mathbf{x}_k + B_k\mathbf{u}_k \tag{2.40}$$
$$\mathbf{y}_k = Cx_k + D_k\mathbf{u}_k \tag{2.41}$$

where:

- $\mathbf{x}_k \in \mathbb{R}^n$ is the state vector of size n at time step k.
- $\mathbf{u}_k \in \mathbb{R}^m$ is the input vector of size m at time step k.
- $\mathbf{y}_k \in \mathbb{R}^p$ is the output vector of size p at time step k.
- $A_k \in \mathbb{R}^{n\times n}$ is the system matrix at time step k, representing how the state variables change with respect to time in response to the system dynamics.
- $B_k \in \mathbb{R}^{n\times m}$ is the input matrix at time step k, representing the influence of external forces on the system dynamics.
- $C_k \in \mathbb{R}^{p\times n}$ is the output matrix at time step k, relating the state vector to the output vector.
- $D_k \in \mathbb{R}^{p\times m}$ is the transmission matrix at time step k, representing the direct influence of the input vector on the output vector.

Re-examining the spring-mass system in Fig. 2.7 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 $\mathbf{x}(k)$ of the mass from its equilibrium position, defined as the position of the mass along the x-axis at time k, and 2) $v(k)$, 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 $\mathbf{x} = [x(k), v(k)]$. The state equations for the spring-mass system are then scaled by the discretization rate Δt of the continuous form at time step k, meaning that the state equation can be formulated in matrix form as:

$$\begin{bmatrix} x_{k+1} \\ v_{k+1} \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & \Delta t \\ -\frac{p\Delta t}{m} & 1 - \frac{c\Delta t}{m} \end{bmatrix}}_{\mathrm{A}_k} \begin{bmatrix} x_k \\ v_k \end{bmatrix} + \underbrace{\begin{bmatrix} 0 \\ \frac{\Delta t}{m} \end{bmatrix}}_{\mathrm{B}_k} u_k, \tag{2.42}$$

where:

- $\mathbf{x}_k \in \mathbb{R}^n$ is the state vector of size n at time step k.
- $\mathbf{u}_k$ is the external force or the input vector of size m at time step t.
- $A_k \in \mathbb{R}^{n \times n}$ is the state transition matrix, representing how the state variables change with respect to time in response to the system dynamics.
- $B_k \in \mathbb{R}^{n \times m}$ is the input matrix at time step k, representing the influence of external forces on the system dynamics.
- Δt is the discretization rate or step size of discretization of the continuous signal.
- p and m are the spring constant and mass, respectively. Previously the spring constant was denoted as k, however since time steps are not denoted as k we accommodate for this arbitrary change.

The matrices A_k, B_k, C_k, and D_k vary with time k in discrete-time SSMs, modelling how the dynamic system evolves over time steps.

2.2.3 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 scalar case with the state equation as follows at every time step t:

$$x_{t+1} = ax_t + bu_t \tag{2.43}$$

Expanding Eq. (2.43) over multiple time steps:

$$\begin{aligned} x_{t+2} &= ax_{t+1} + bu_{t+1} && (2.44)\\ &= a(ax_t + bu_t) + bu_{t+1} && (2.45)\\ &= a^2x_t + abu_t + bu_{t+1} && (2.46) \end{aligned}$$

$$\begin{aligned} x_{t+3} &= ax_{t+2} + bu_{t+2} && (2.47)\\ &= a(a^2x_t + abu_t + bu_{t+1}) + bu_{t+2} && (2.48)\\ &= a^3x_t + a^2bu_t + abu_{t+1} + bu_{t+2} && (2.49) \end{aligned}$$

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

$$P(x) = x_{t+n} = a^n x_k + a^{n-1}bu_t + a^{n-2}bu_{t+1} + \cdots + abu_{t+n-2} + bu_{t+n-1} \tag{2.50}$$

2.2.4 High-Order Polynomial Projection Operator

High Order Polynomial Projection Operator (HiPPO) [53] utilizes the convenient formulation of an SSM as a polynomial as shown in Eq. 2.50, to project 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 depicted in Fig. 2.8 can then be summarised as follows: 1) for any function f, 2) at every time step t, HiPPO projects $f(t)$ onto $g(t)$ a sub-space of orthogonal polynomials with respect to a measure $\mu(t)$, 3) for which the corresponding polynomial coefficients $\mathbf{c}_t \in \mathbf{R}^N$ represent the compressed history of f, 4) leading to a discrete time-dynamic recurrent relationship between consecutive time steps by formulating the problem as an SSM:

$$\mathbf{c}_{t+1} = A_t \mathbf{c}_t + B_t \mathbf{u}_t \tag{2.51}$$

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. 2.40 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 SSM Eq. 2.40.
- $A(t) \in \mathbb{R}^{n \times n}$ 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.

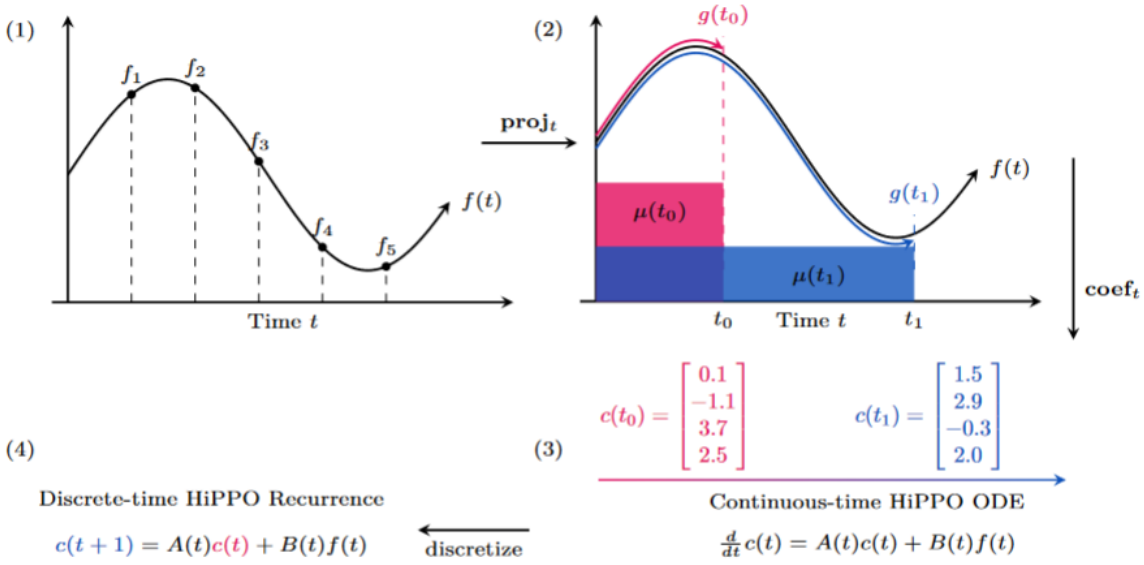


Figure 2.8: General HiPPO framework. (1) For any function f, (2) at every time t there is a projection $g(t)$ of f onto the space of polynomials, with respect to a measure $\mu(t)$ weighing the past. (3) For a chosen basis, the corresponding coefficients $c(t) \in \mathbb{R}^N$ represent a compressed of the history of f satisfying linear dynamics. (4) Discretizing the dynamics yields an efficient closed-form recurrence for online compression of time series $(f_k)_{k \in \mathbb{N}}$.

Note that the formulation in Eq. (2.51) 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 gradient problems. That is, HiPPO can be used as a gating mechanism for any RNN, for example an LSTM as shown in Fig. 2.9. 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 simple 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 scala 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 n-th order orthogonal polynomial. Formally this recurrent relationship can be expressed as:

$$\mathbf{h}_t \in \mathbb{R}^N = \tau(\mathbf{h}_{t-1}, [\mathbf{c}_{t-1}, x_t]) \tag{2.52}$$
$$\mathbf{f}_t \in \mathbb{R}^1 = f_W(\mathbf{h}_t) \tag{2.53}$$
$$\mathbf{c}_t \in \mathbb{R}^N = hippo_t(f_t) \tag{2.54}$$
$$= A_t \mathbf{c}_{t-1} + B_t f_t, \tag{2.55}$$

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 time step t

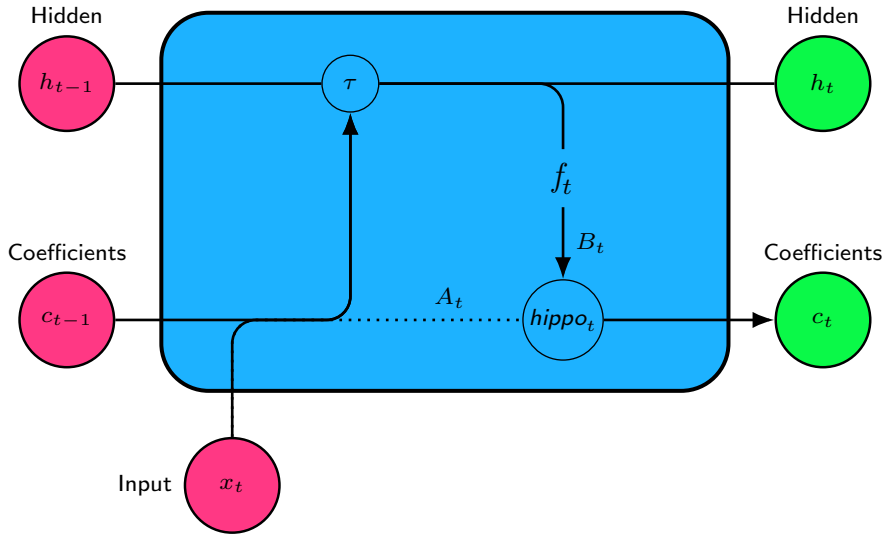


Figure 2.9: Gating mechanism of a HiPPO-LSTM cell. The cell takes as input the input at the current time step t, the hidden state at the previous time step $t-1$ and cell state of the previous time step $t-1$, returning the hidden state and cell state of the current time step.

Chapter 3

Method

We use a YoloV4 [23] framework for both still-image and video object detection. In particular we train YoloV4 on still-image object detection, using the MS COCO2017 [54] dataset. The resulting model is then used to fine-tune three models: 1) YoloV4, 2) YoloV4 + UR-LSTM (UrYoloV4) and 3) YoloV4 + HippoLSTM (HoloV4) on video object detection, using the ImageNet VID 2015 [55] dataset.

3.1 Sequence Models

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 [75]. 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 [53]. Since they are Legendre 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:

$$\mathbf{c}_{t+1} = -\frac{1}{t} A_t \mathbf{c}_t + \frac{1}{t} B_t \mathbf{u}_t \tag{3.1}$$

$$A = \begin{cases} \sqrt{(2n+1)(2t+1)} & \text{if } n > t \\ n+1 & \text{if } n = t \\ 0 & \text{if } n < t \end{cases} \tag{3.2}$$

$$B = \sqrt{2n+1} \tag{3.3}$$

where the corresponding A and B matrices are defined as above and scaled with respect to time, resulting in a set of scaled Legendre or LegS polynomials. Continuous time A and B matrices are discretized using the bilinear transform [74]. For more details, we refer to [53].

3.2 YoloV4

YoloV4 [23] 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 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 3.1.

3.2.1 Architecture

YoloV4 utilizes a network that is called Darknet-53, which consists of 53 convolutional layers pre-trained on ImageNet classification, followed by co-adaptation layers, a neck consisting and a prediction head consisting of 2 convolutional layers making predictions for each one of the three scales: $76 \times 76\times$, $38 \times 38\times$ and 19×19 [23]. A detailed depiction of the network architecture can be seen in Figure 3.1 with each component being labeled and sketched out. We deviate slightly from the original architecture by using an EfficientNetS [76] 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 (SPP) [77] component, a Path Aggregation Net (PANet) [78] and a final prediction head. By passing the tensors from the co-adaptation block into the neck, the SPP 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 passed onto each of the responsible prediction heads, resulting in three output tensors, one for each of the three scales.

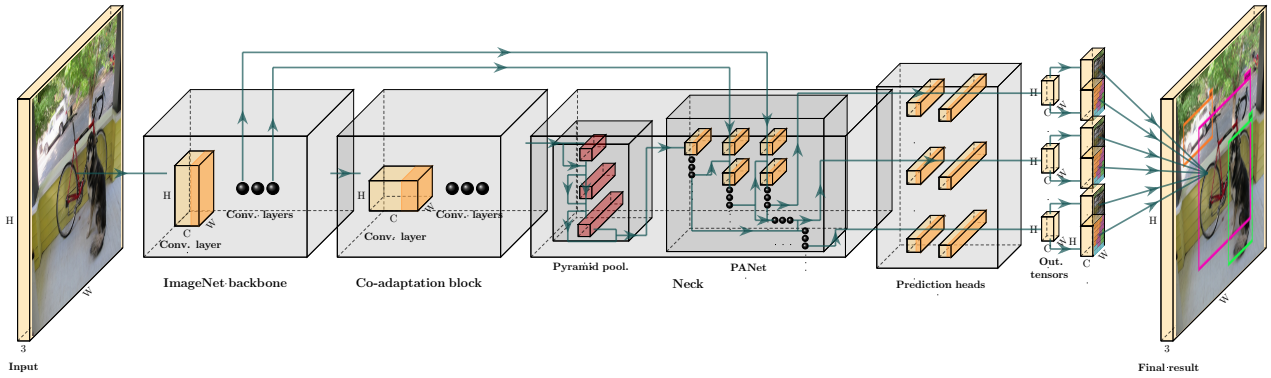


Figure 3.1: YoloV4 workflow for video object detection. The linear layer from YoloV1 has been replaced by three convolutional prediction heads, each making predictions at different grid sizes or scales. The network consists of four parts: 1) an ImageNet backbone, 2) a co-adaptation block, 3) a neck and 4) a predictions head, making predictions at three scales (76, 38, 19).

3.2.2 Loss

YoloV4 formalizes the optimization landscape through multiple sums of squared errors (SSE) for localization (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 SSE terms are easy to optimize, they do not perfectly align with the goal of maximizing performance metrics. These metrics, such as mean average precision (mAP), depend on the intersection over union (IoU) between true and predicted bounding boxes. Multiple SSE terms will weigh different localization objectives equally to each other, causing the model to weigh positional errors (i.e., x, y) equal to errors in size (i.e., height and width). This equal weighing can result in situations where the model minimizes errors in height and width without adequately addressing errors in position or vice versa. Consequently, this can reduce the IoU between true and predicted bounding boxes, leading to a suboptimal solution. In addition to this, since many of the grids in the images will not contain any object, the confidence score of these cells 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 address these two issues, the loss components are weighted differently, using parameters $\lambda_{class} = 1$, $\lambda_{box} = 8$, $\lambda_{noobj} = 4$ and $\lambda_{obj} = 2$ for example.

This, however, does not resolve the issue that SSE weighs errors in large boxes and small boxes equally. Since loss should reflect that small deviations in large boxes are less important than small deviations in small boxes, this is problematic. For example, 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. 3.4 illustrates that the first two loss components are part of the 1) localization 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 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 IoU for any predictors in that grid cell i (see Fig. 3.2). Consequently, there is only a loss if there is an object in cell i given bounding box j is responsible.

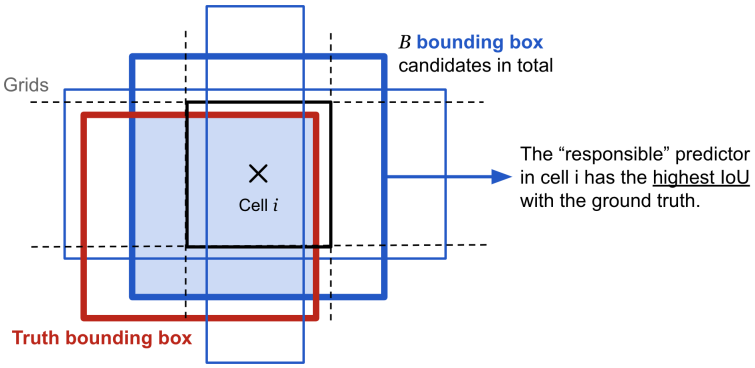


Figure 3.2: Given cell i shown as a black square, the model proposes B number of bounding box candidates. The bounding box candidate that has the highest intersection over union with the ground truth bounding box is the responsible predictor.

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 accomplished by weighing 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 $\hat{p}_i(c)$.

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

$$
\begin{aligned}
&+ \underbrace{\lambda_{\text{noobj}}}_{\text{constant}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \underbrace{I_{ij}^{\text{noobj}}}_{\text{identity}} \underbrace{\left(C_{ij} - \hat{C}_{ij}\right)^2}_{\substack{\text{error term} \\ \text{confidence scores}}} \\
&+ \underbrace{\lambda_{\text{class}}}_{\text{constant}} \sum_{i=0}^{S^2} I_i^{\text{obj}} \sum_{c \in \text{classes}} \underbrace{-\left(\hat{p}_i(c) \log(p_i(c)) + (1 - \hat{p}_i(c)) \log(1 - p_i(c))\right)}_{\substack{\text{bce error term} \\ \text{class probabilities}}},
\end{aligned}
\tag{3.4}
$$

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.

3.3 Datasets

3.3.1 Modified National Institute of Standards and Technology Database

The modified National Institute of Standards and Technology database (MNIST) [75] 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.

3.3.2 Microsoft Common Objects in Context Dataset

The Microsoft Common Objects in Context (MS COCO) dataset [54] 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 test examples, annotated with bounding boxes, per-instance segmentation mask and class categories for 80 object categories.

3.3.3 ImageNet Video Dataset

ImageNet Video (VID) dataset [55] is a large-scale public dataset for video object detection that 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.

Chapter 4

Experiments

4.1 Sequence Models

Sequence models are validated on the sequential MNIST task [75], 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 image are flattened into a single vector, each index now representing a single pixel 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. We examine models like RNNs, vanilla RNN, GRU, LSTM, uRNN, and HiPPO because they are designed to capture long-term dependencies in sequential data. Each model introduces different mechanisms to maintain and update memory, allowing them to address the challenge of processing long sequences, such as the sequential MNIST task, with varying levels of efficiency and performance. These comparisons help identify models that are best suited for tasks requiring memory over long time steps. Each model is trained 5 times. A single hidden to hidden layer was used for each model, defining shallow RNNs in terms of their depth. The mean and standard deviation is computed across 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 4.1.

Table 4.1: 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

4.1.1 Results

Results for the mean accuracy and standard deviation, computed after 5 repetitions on the sequential MNIST task can be seen in Table 4.2. The values of the highest performing models are highlighted in boldface. Graphically, loss can be examined in Fig. 4.1 and accuracy in Fig. 4.2. From the loss in Fig. 4.1, we observe that a vanilla RNN is particularly prone to the exploding and vanishing gradient problem as visible by a large increase in mean loss and the shaded area in blue. 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 4.2: 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**

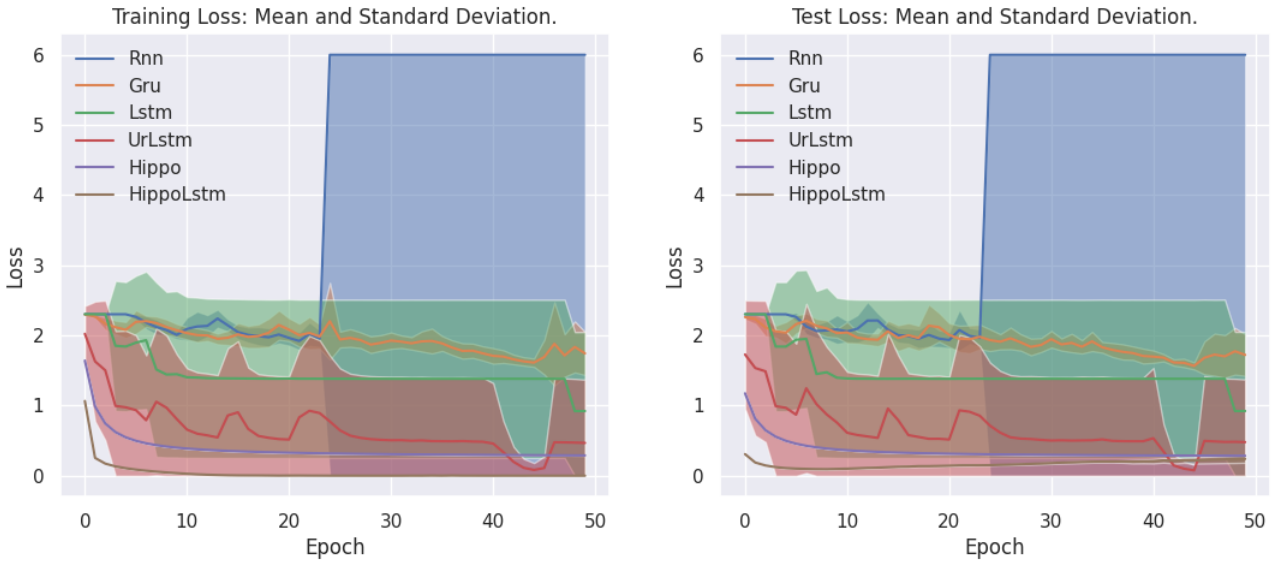


Figure 4.1: Train vs. test loss as a function of epoch across different recurrent neural networks with the standard deviation represented as shaded areas. Loss values are bounded above at 6.0 for graphical illustration.

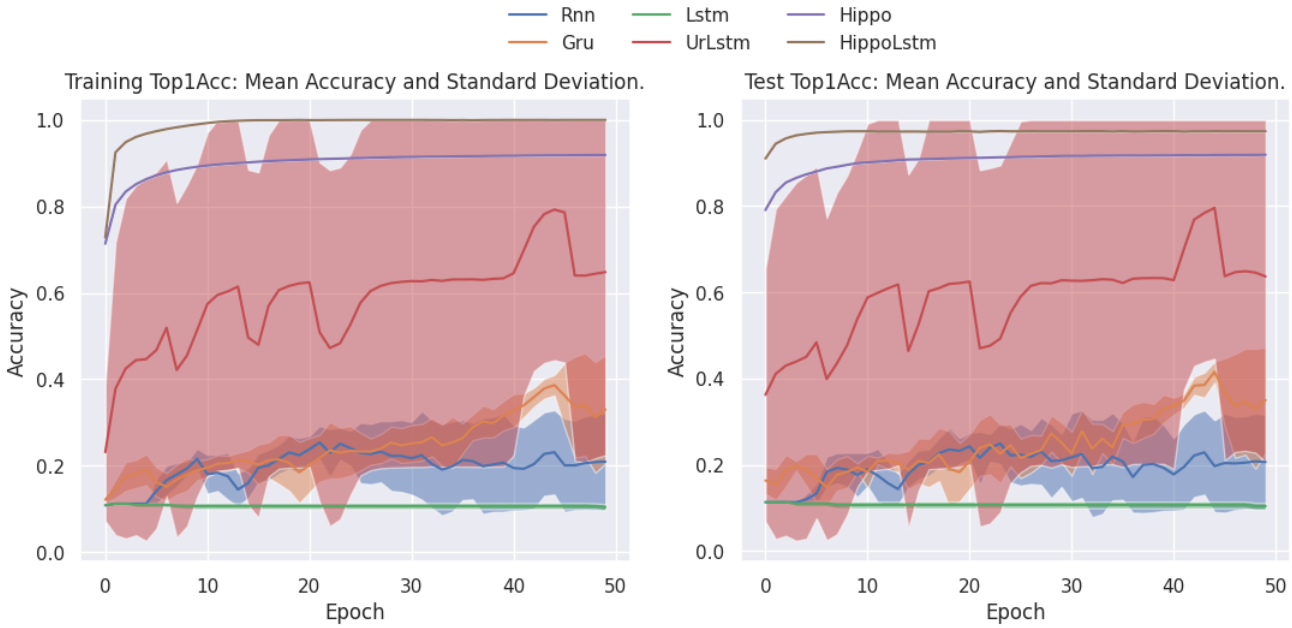


Figure 4.2: Train vs. test accuracy as a function of epoch across different recurrent neural networks with the standard deviation represented as shaded areas.

4.2 Still-Image Object Detection

Since the still-image detector is used to initialize the video-object detector in subsequent experiments, multiple repetitions 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 increase 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. Augmentations are applied with a probability of p. That is, each individual image has a probability p of being augmented. 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 4.3.

Table 4.3: 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

4.2.1 Results

The results for YoloV4 on the MS COCO 2017 dataset can be seen in Table 4.4, where class accuracy, object accuracy, no object accuracy, mAP@0.5, recall and precision are depicted. An example of the trained detection model on a test image can be seen in 4.3. 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 4.4: 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

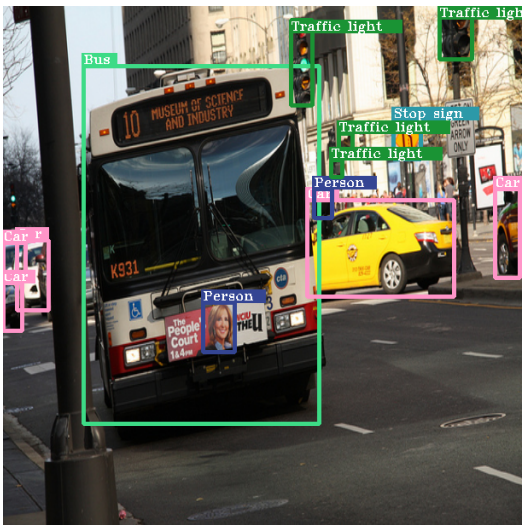


Figure 4.3: YoloV4 detections on a test still-image from the MS COCO 2017 dataset.

4.3 Video Object Detection

Since YoloV4 makes predictions at 3 scales (i.e., 76, 38 and 19), the network's final feature map rapidly increases 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 for the sequence models we 1) only process 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. Each individual tensor for x, y, w, h has a length of 1083 for each time step t.

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 padding the image vector by repeating the sequence of earlier frames.

The input image is resized to have a square height and width of 608. 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) grey-scaling: $p_g = 0.1$, 3) blurring $p_b = 0.1$, 5) CLAHE $p_{clahe} = 0.1$, 6) random channel shuffle: $p_{cs} = 0.1$ and 7) posterize: $p_p = 0.1$.

For the Yolo model that utilizes an UR-LSTM (UrYolo), a deep UR-LSTM with 6 hidden to hidden layers and a hidden state size of 1024 for each layer is used. Less lead to performance degradation and more did not lead to performance increases during fine-tuning. For the Yolo variant that utilizes a HiPPO-LSTM (Holo), stacking layers on-top of each other did not result in noticeable performance difference, so a single layer with 1088 coefficients was used. Thus, the HiPPO-LSTM module in Holo is around 5.6 times smaller than the UR-LSTM in UrYolo.

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. As model confidence can decrease when using sequence models, this can negatively impact video object detection performance by causing potentially correct detections to be discarded, even if they accurately identify objects in the video. Therefore, we examine mAP by increasing confidence in steps of 0.01 from 0.0 to 1.0 additionally.

Table 4.5: 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	$1e-3$	0.0	0.937	200	32	one cycle	Adam	1.0	2.0	4.0	10.0
UrYoloV4-608	$1e-3$	0.0	0.937	200	32	one cycle	Adam	1.0	2.0	4.0	10.0
HoloV4-608	$1e-3$	0.0	0.937	200	32	one cycle	Adam	1.0	2.0	4.0	10.0

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

4.3.1 Results

Training and test loss as a function of epochs can be examined in Fig. 4.4. Overall, for both training and test loss, Yolo results in the lowest loss value compared to UrYolo and Holo. Standard deviations are shaded and colored for each model, giving an indication of stability. Large standard deviations reflect larger spread of values between multiple runs and therefore a less stable solution. Smaller standard deviations reflect smaller spread and a more stable solution across multiple runs. Visible are oscillation in the size of the standard deviation as represented by the shaded areas. Larger standard deviations are especially visible in the sequence models Holo and UrYolo during earlier training phases. Yolo has the smallest standard deviation during training as epochs increase. Holo having particularly small standard deviations during epoch 75 to 120 and UrYolo for epoch 150 to

175. However, all models converge to a smaller standard deviation as seen by the small, shaded area on the training data. Standard deviations as a function of epochs are larger for all models on the test data. Larger oscillations in loss are also visible, as the model needs to generalize, which is expected. Even though Yolo has the smallest standard deviation and smallest loss on the training loss, standard deviations are larger. The test loss is still the smallest, followed by UrYolo and Holo. Furthermore, UrYolo and Holo converge to smaller standard deviation than Yolo, indicating a more stable solution than Yolo across multiple runs.

Evaluation metrics for both the training and test dataset at the final epoch with a confidence value of 0.8 are shown in Table 4.6 and 4.7 respectively. The mean and standard deviation is computed across 2 repetitions. With a high confidence of 0.8, the Yolo baseline has the highest mAP @0.5 = 0.345 on the test data with a standard deviation 0.03. The UrYolo has a mAP@0.5 = 0.313 and a standard deviation of 0.037. Holo has the smallest standard deviation 0.016, but also the lowest mAP@0.5 = 0.228. Graphically mAP as a function of epochs on both the training and test data can be examined in Fig. 4.5. At a confidence threshold of 0.8, Yolo has a larger mAP across all epochs compared to other models for both training and test data. Standard deviations on the training data are also smaller for Yolo, where standard deviations oscillate with a higher frequency as a function of epochs for UrYolo and Holo as visible by the shaded areas. Larger standard deviations and oscillation in mAP on the test data compared to the training data are expected and visible for all models as models try to generalize. Spread in mAP increases as a result, where shaded areas between Yolo and UrYolo can intersect particularly at later epochs (i.e., at approximately epoch 155 to 168 and near 200). Holo's standard deviations do not intersect with Yolo, however can intersect with UrYolo before epochs smaller than 170 converging to the lowest mAP on the test data.

Recall, which the model's ability to correctly detect all relevant objects in an image, measuring how many of the actual objects were successfully identified and precision, which measures the accuracy of the detected objects, indicating how many of the objects that the model identified were correctly classified are shown in column 9 to 12 in Table 4.6 and Table 4.7. For the models under consideration on the test data, UrYolo has the highest precision of 0.786 with a standard deviation of 0.082, followed by Yolo with a precision of 0.697 and a standard deviation of 0.0480, and Holo with a precision of 0.689 and standard deviation of 0.027. With respect to recall, Yolo has the highest recall of 0.355 with standard deviation 0.033, followed by UrYolo with a recall of 0.317 and standard deviation of 0.039, and Holo with a recall of 0.235 and standard deviation of 0.011.

Since confidence can decrease over prediction when using sequence models, mAP@0.5 as a function of different confidence values can be seen in Fig. 4.6. At confidence values approximately below 0.5 all models show no change in mAP, effectively staying constant. After this, mAP for all models changes. While Holo overall converges to the lowest mAP, there is a small increase at around a confidence value of 0.50, decreasing at approximately 0.57 again. UrYolo on the other hand increases in mAP approximately after 0.5, developing a higher mAP than the Yolo baseline and reaching its peak at 0.57 before decreasing again. Holo and in particular UrYolo show slight improvements in mAP at certain confidence intervals. In contrast, Yolo performance decreases as a function of increasing confidence values. Because higher confidence threshold values filter out predictions, meaning that higher values lead to less predictions and smaller values to more predictions, it is not possible to set confidence to zero as there will be too many bounding boxes than objects visible in an image. Similarly, it is not advised to set the confidence value too high, as there will be less or no predictions than objects in an image.

Therefore, we examine the number of predictions across each model at confidence equal to 0.57 in Tab. 4.8. UrYolo leads to the highest mAP of 0.40 and the sparsest number of 835 predictions compared to Holo and Yolo. Yolo has a second highest mAP of 0.369 with 1027 predictions. Holo the lowest mAP of 0.315 with the largest number of 1142 predictions. With 1023 true bounding boxes in the test dataset, Yolo and especially Holo lead to too many predictions, unlike UrYOLO which makes 188 less than the target number, but still reaches a higher mAP than the other two models. Taken spread across predictions in terms of standard deviation into account, Yolo has the smallest standard deviation 10, followed by UrYolo 57 and Holo with 109. Re-examining recall in column 11 to 12 in Table 4.8, Yolo has the highest recall of 0.661 with a standard deviation of 0.002. UrYolo has the second highest recall of 0.66 with a standard deviation of 0.072. Holo has the smallest recall of 0.467 with a standard deviation of 0.009. Inspecting recall in column 13 to 14, UrYolo now has the highest recall of 0.42 with standard deviation 0.037. Yolo has the second largest recall of 0.383 with standard deviation of 0.037. Holo has the smallest recall of 0.338 with a standard deviation of 0.024.

Consequently, when comparing test precision at confidence 0.8 with a confidence at 0.57, precision across all models decrease. The difference in decrease in terms of precision between these two confidence values for the Yolo model is equal to $0.697 - 0.661 = 0.036$. The decrease in precision between these two confidence values for the UrYolo model is $0.786 - 0.66 = 0.126$ and for Holo $0.689 - 0.467 = 0.222$. Therefore, the decrease from largest decrease to smallest is: Holo, UrYolo and then Yolo. While precision decreased across all models, recall increased. UrYolo now has the highest recall of 0.42 with a standard deviation of 0.037. Yolo has the second largest recall of 0.383 with standard deviation 0.037. Holo has the smallest recall of 0.338 a standard deviation of 0.024. Overall, UrYolo recall increased by $0.42 - 0.317 = 0.103$, Yolo by $0.383 - 0.355 = 0.028$ and Holo by $0.338 - 0.235 = 0.103$. Thus, UrYolo and Holo show the largest increase in recall and Yolo the smallest.

Figure 4.4: Train vs. test loss as a function of epochs across different recurrent Yolo networks with the standard deviation represented as shaded areas.

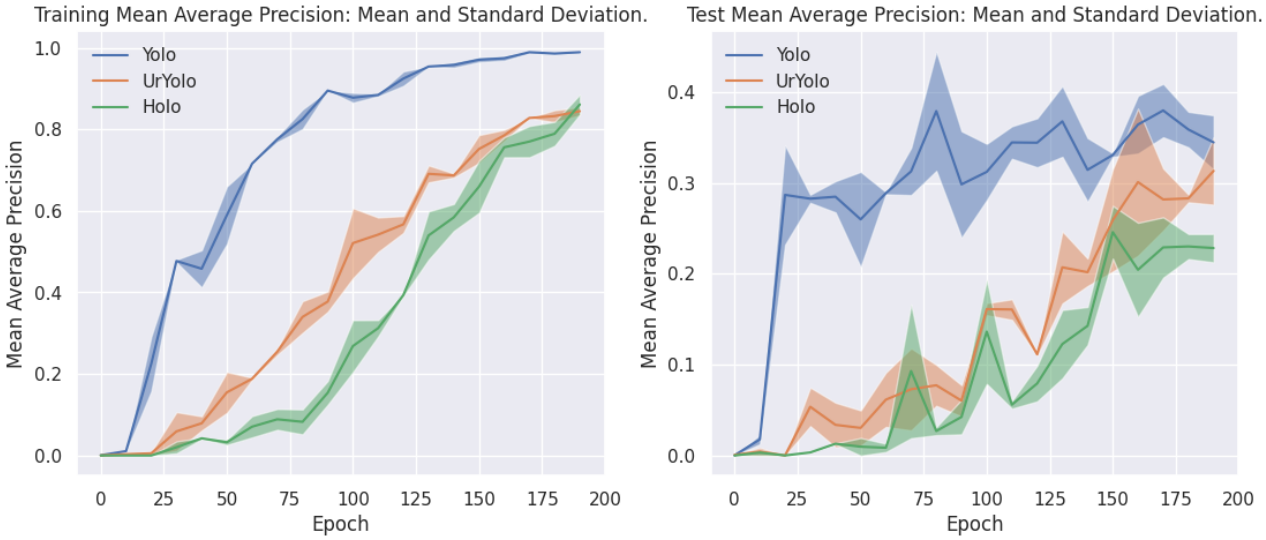


Figure 4.5: Train vs. test mean average precision as a function of epochs for different recurrent Yolo networks with the standard deviation represented as shaded areas. Values are rounded to 3 decimal points.

Table 4.6: Metrics on the training dataset: Mean mAP@0.5 and standard deviation (SD) across 2 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	**0.989**	0.0	0.998	0.002	0.973	0.019	0.999	0.0	0.987	0.002	0.991	0.001
Ur-YoloV4	**0.844**	0.01	1.0	0.0	0.692	.044	0.999	0.0	0.995	0.0	0.845	0.01
HoloV4	**0.861**	0.0245	1.0	0.0	0.779	0.056	0.999	0.0	0.983	0.002	0.863	0.024

Table 4.7: Metrics on the test dataset: Mean mAP@0.5 and standard deviation (SD) across 2 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	**0.345**	0.03	0.255	0.028	0.243	0.059	0.999	0.0	**0.697**	0.048	**0.355**	0.033
Ur-YoloV4	**0.313**	0.0371	0.235	0.01	0.171	0.024	0.999	0.0	**0.786**	0.082	**0.317**	0.039
HoloV4	**0.228**	0.016	0.156	0.049	0.161	0.034	0.999	0.0	**0.689**	0.027	**0.235**	0.011

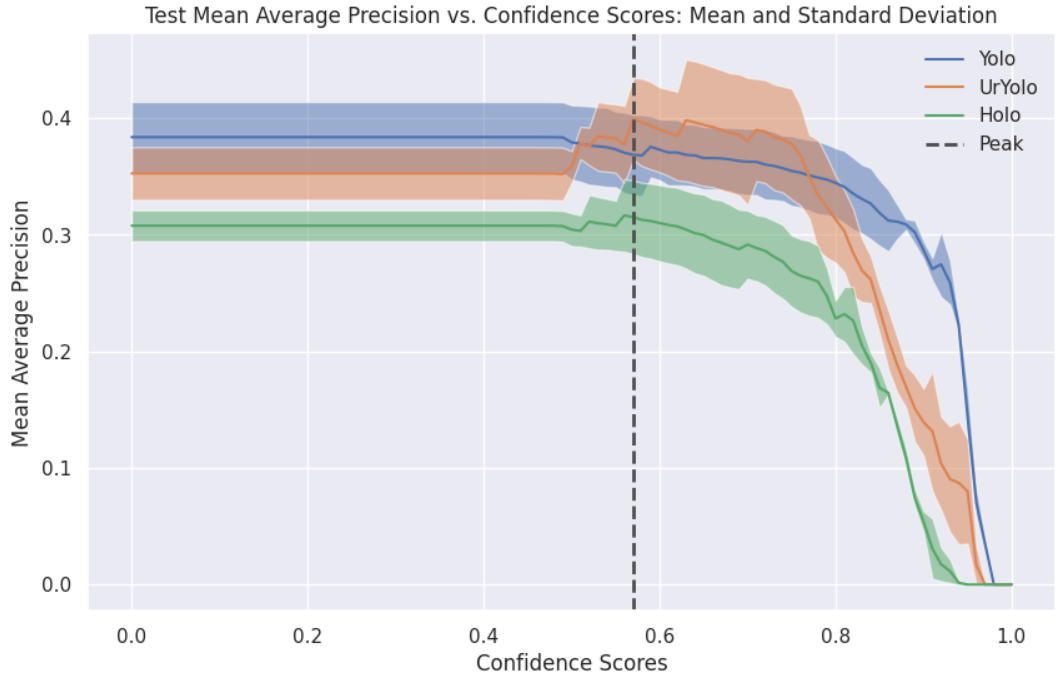


Figure 4.6: Test mean average precision as a function of different confidence score thresholds. Predictions below a confidence score are filtered out. That is, with confidence score of 0.4 predictions with a confidence score below 0.4 are omitted. Dotted line repents the mAP at a confidence threshold of 0.57, which is the peak or point of highest mAP of the UrYolo model.

Table 4.8: Metrics on the test dataset at confidence 0.57: Mean mAP@0.5 and standard deviation (SD) across 2 repetitions on the ImageNet VID dataset after training. Values are rounded to 3 decimal points. The number of bounding box predictions are included, where the true number of bounding boxes in the test data is 1023.

Model	Mean_{preds}	SD_{preds}	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	**1027**	10	**0.369**	0.035	0.689	0.007	0.701	0.007	0.998	0.0	**0.661**	0.002	**0.383**	0.037
Ur-YoloV4	**835**	57	**0.40**	0.035	0.681	0.0161	0.523	0.023	0.997	0.0	**0.66**	0.072	**0.42**	0.037
HoloV4	**1142**	109	**0.315**	0.032	0.635	0.013	0.577	0.053	0.995	0.0	**0.467**	0.009	**0.338**	0.024

Chapter 5

Discussion

In this thesis, we explored three main areas of deep learning, focusing on validating different models across tasks that included sequence processing and object detection. First, we evaluated the performance of sequence models, specifically Recurrent Neural Networks (RNNs) and State Space Models (SSMs) on the Sequential MNIST dataset. Our analysis aimed to assess these models' capabilities in handling long-term dependencies in sequential data. Second, we applied YoloV4 [23] for still-image detection, evaluating its effectiveness in accurately identifying objects within static visual scenes. Finally, we extended our YOLOv4 object detection approach to the video domain by incorporating a combination of YOLOv4, UR-LSTM, and HiPPO-LSTM models to perform video object detection. We called these models Yolo, UrYolo and Holo respectively. This approach aimed to capture temporal dependencies and improve detection performance in dynamic environments. Collectively, these tasks provide a comprehensive evaluation of sequence model performance across different data types, contributing to our understanding of their strengths and limitations in real-world scenarios.

In the following subsections, we reiterate the key points of the results from each task, interpreting their implications within the broader context of deep learning and object detection research. This includes a detailed comparison of the sequence models (RNNs and SSMs) on the Sequential MNIST task, an analysis of YOLOv4's performance in still-image detection, and an exploration of the video object detection capabilities using the combined approaches of YOLOv4, UR-LSTM, and HiPPO-LSTM. These interpretations will highlight the significance of our findings, address observed limitations and suggest potential improvements for future research to build on our results.

Sequence Models 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 (i.e. vanilla RNN, LSTM, GRU or RNN) have an accuracy below 40.0%. These results are also reflected in the standard deviations in Table 4.2, where the HiPPO-LSTM and HiPPO have the smallest standard deviation. HiPPO introduces 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. This instability can be examined visually in Fig. 4.2 by the shaded areas. Examining the solution in terms of the loss in Fig. 4.1 and 4.2 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 4.2, loss as depicted in Fig. 4.1 is still lower than for regular RNN variants. Particularly striking is the substantial increase of the training and test loss of the vanilla RNN in Fig. 4.1 shown in blue. As the vanilla RNN does not introduce any gating mechanism to control the exponential increase (or decay) of gradients, both training and test loss provide an ideal example of the exploding and vanishing gradient problem. This causes loss to spike to exceptionally large values and to zero abruptly. GRU and LSTM are more well behaved, as loss values do not increase to large values or decrease to zero. However, the LSTM has larger standard deviations than GRU, reflecting a less stable solution.

That is, even though the UR-LSTM has the largest standard deviation, meaning that the model converges to a less stable solution, it is to be preferred compared to other regular RNN variants. In some cases, as visible by the shaded area in red for the test accuracy in Fig. 4.2, 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. Given 5 repetitions, its mean accuracy is however still higher than regular RNN variants like the vanilla RNN, GRU or LSTM.

Overall, this set of experiments illustrates the incremental improvement gating mechanisms have made over the years in their ability to memorize long context and alleviate the vanishing and exploding gradient problem during optimization. Orthogonal projections as utilized by HiPPO seem to address the vanishing and exploding gradient issue, leading to both stable solutions and higher accuracy on a long context ask.

Still-Image Detection 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 uses the same backbone. From Table 4.4, we see 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 predicting no objects where there are no objects, but not as good at predicting 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 weigh each loss component differently, have a higher emphasis on λ_{noobj}, limiting the number of false positives (i.e., predicting an object 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 are reflected 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%, meaning 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).

Limitations One limitation of our approach is that, although we know the true bounding box locations, we currently do not leverage segmentation masks as a supervised signal. That is, by using true bounding box locations, a segmentation mask that identifies where and what objects are present at each scale can be computed, which could then be used to fine-tune an additional scale dependent segmentation task. This adjustment can improve performance, as it would provide the model with more precise spatial information, leading to a better ability to distinguish between object and no-object regions effectively. Naturally, this approach has been implemented in YoloV5 [79]. As a result, YoloV5 achieves improved accuracy by dynamically adjusting to object sizes and refining segmentation mask predictions across different scales.

Video Object Detection Lastly, we find that giving an object detector access to temporal information through bounding box trajectories using Yolo can lead 1) sparse predictions over certain confidence threshold intervals and 2) to performance increases compared to baseline. This is particularly true for UrYolo, which increases mAP by 3.1%, while also making 192 less predictions than Yolo. Considering, that there are 1023 true bounding box predictions in the test data, UrYolo makes 188 less predictions, but increases model performance at a confidence threshold of 0.57 as shown in Table 4.8. This would suggest that the UR-LSTM recurrence regularizes the number of predictions but can leverage previous context to improve the intersection over union when making predictions to increase bounding box accuracy.

Holo on the other hand does not share this behavior. Overall Holo leads to a decrease of 5.4% in terms of mAP and an increase of 115 bounding box predictions compared to Yolo baseline. This performance degradation is also reflected in the training and test loss as shown in Fig. 4.4. Holo train and test loss is the highest among all models. This, however, does not hold when examining training mAP, where Holo outperforms UrYolo, but still has the lowest mAP among all models as it is not able to generalize as well to the test data (see Fig. 4.5). While surprising given the excellent performance of the HiPPO-LSTM on the sequential MNIST task, HiPPO was not designed to process elements in a vector in parallel. That is, given an input vector x of length L, HiPPO unrolls sequentially over each input x_t for $t \in \{1, 2, 3, \ldots, L$, mapping the input 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 the entire vector of length L causes a great loss of information. 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 in our fine-tuning phase. That is, HiPPO needs to sequentially process individual elements in the bounding box vector of length L, one vector for each x, y, w, h coordinate, one by one, before it can pass the coefficient that represent the context, from one frame to the next. As such, HiPPO was not designed for processing entire vectors at a given time step, requiring the problem to be formulated as a long-context task, leading to performance degradations and the loss of real-time object detection capabilities. Even though this is the case,

it should be noted that the HiPPO-LSTM module in Holo has a size of 1088 and the UR-LSTM module in UrYolo has a size of 6144, meaning it is approximately 5.6 times smaller. A UR-LSTM module of the same size should not be able to reach the same performance, which we observed during fine-tuning.

Overall UrYolo reaches the highest mAP of 40.0%, Yolo the second highest mAP of 36.9% and Holo the lowest mAP of 31.5% at confidence 0.57. This does not hold for a higher confidence at 0.8, where Yolo has the highest performance with a mAP of 34.5%, UrYolo with a mAP of 31.3% and Holo with a mAP of 22.8%. However, because higher confidence threshold values filter out more predictions, meaning that higher values lead to less predictions and smaller values to more predictions, it is not advised to set confidence to zero or too high (i.e., 1.0). This is because, if the confidence is too low, there will be more predictions than objects in the image, cluttering the image with redundant bounding boxes. For a similar reason, if confidence is set too high objects within the image will be missed. This relationship between different confidence threshold values relationship of confidence on mAP can be examined in Fig. 4.6. The mAP reaches 0.0% when confidence is 1.0 and increases as it is lowered to 0.0. Considering that Yolo makes 1027 and Holo 1142 compared to 1023 target predictions, we find a confidence value of 0.57 to work quite well. Since Yolo predictions have a standard deviation of 10 and Holo of 109, they can fall below or outside the number of true predictions. Given that 0.57 was the smallest floating value in steps of 0.01 that lead to the closest number of predictions to the true number of predictions, this is a reasonable value. Since mAP decreased before the 0.8 threshold, a confidence value of 0.8 is too high.

Relaxing the confidence threshold does not only influences the number of predictions and mAP, but also has an impact on precision and recall. Precision is the proportion of correctly identified objects among all predicted objects and recall is the proportion of correctly identified objects out of all the actual objects present in the image. Lowering the confidence threshold from 0.8 to 0.57 causes precision to decrease while recall increases. Specifically, precision decreases by 3.6% from 69.7% to 66.1%, 12.6% from 78.6% to 66.0%, and 22.2% from 68.9% to 46.7% for Yolo, UrYolo, and Holo, respectively, while recall increases by 2.8% from 35.5% to 38.3%, 10.3% from 31.7% to 42.0%, and 10.3% from 23.5% to 33.8% for Yolo, UrYolo and Holo in that order. This suggests a trade-off between precision and recall. Overall, Yolo shows a modest decrease in precision and a slight increase in recall, indicating that while it becomes slightly more prone to false positives when the threshold is lowered, it still manages to detect more true objects without a dramatic loss in precision. UrYolo experiences a more significant drop in precision but also a substantial gain in recall. This suggests that UrYolo trades off more precision for recall when the confidence threshold is reduced, allowing it to detect more true objects at the expense of a higher rate of false positives. Holo is affected most by the reduction in the confidence threshold, with a dramatic drop in precision and a notable increase in recall. This indicates that Holo struggles with a higher rate of false positives when the threshold is lowered, even though it successfully identifies more true objects. These changes indicate that while reducing the threshold helps each model capture more objects (higher recall), it does so by sacrificing accuracy in those predictions (lower precision), with Holo showing the most notable change in both metrics and Yolo being most robust to it.

Overall, UrYolo stands out as the most effective model when leveraging temporal information, achieving the highest mAP and balancing precision and recall well compared to the baseline. While Yolo performs consistently and robustly across different thresholds, Holo struggles due to its sequential processing limitations, leading to significant performance drops and increased false positives.

Limitations Despite the promising results of UrYolo in utilizing temporal information, several limitations remain. The biggest limitation is that the HiPPO framework is not designed to process an entire input vector in parallel. HiPPO formulates the problem as a long sequence problem. That is, given a sequence of images denoted as $\{I_t\}_{t=1}^{T}$, where T represents the total number of frames in the video, let the bounding box prediction for each image I_t at time step t be represented by four separate vectors of length K: $\mathbf{x}_t$, $\mathbf{y}_t$, $\mathbf{w}_t$, and $\mathbf{h}_t$, corresponding to the x-coordinate, y-coordinate, width, and height of the bounding box, respectively. Instead of processing an entire bounding box vector in parallel, HiPPO must iterate through each element $k = 1, 2, \ldots, K$ of the vectors one by one for each k. This approach transforms the processing of bounding box predictions into a long, flattened sequence of values over time, such as $\{x_1, x_2, \ldots, x_K\}$ for the x coordinate of the bounding box centers for example.

UrYolo, on the other hand, utilizes a UR-LSTM that can process all elements $k = 1, 2, \ldots, K$ within a single vector simultaneously. Given the same sequence of images $\{I_t\}_{t=1}^{T}$, the UR-LSTM

can processes the entire length K of a single bounding box vector, for example $\mathbf{x}_t$ in parallel at each time step $t = 1, 2, \ldots, T$. This ability to process entire vectors in parallel reduces the complexity, effectively transforming the task into a short sequence problem. As a result, UrYolo can handle sequences more efficiently, unlike HiPPO which needs to iterate over each element in the vector. Therefore, the comparison between UrYolo and Holo is not in favor of Holo, since it is a comparison between a short sequence problem of length T and one that is equal to $T \times K$. In addition to this, since both models assume that the input vectors are flattened into a one-dimensional vector, the spatial arrangement of features from the CNN is arguably lost. 3D Convolutions [80] or other SSM kernels [81] that preserve the spatial and temporal structure of the data, may therefore be more suitable.

Furthermore, since our video object detector utilizes the YoloV4 [23] framework, it can also benefit from utilizing an object segmentation approach. By generating segmentation masks from the true bounding box locations and using them as an additional supervised signal performance can increase, effectively becoming a YoloV5 [79] model. In addition to this, we constrained our dataset by only including videos with a maximum frame count of 80, which limited the diversity of temporal data. This limitation is further exacerbated by our sub-sampling strategy that reduced our frame count by selecting every 10th frame. While this approach improved computational efficiency, it likely resulted in a compounding loss of temporal information, particularly in shorter videos where subtle frame changes are significant. Investigating the model performance on videos with frame counts exceeding 80 and sub-sampling at higher rates may lead to a better understanding of each model's effectiveness on longer sequences.

Chapter 6

Conclusion

Firstly, in this study, we replicated findings from [52; 53] 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 [23] object detector, that shows a slight improvement over an existing implementation that uses the same EfficientNet backbone. This approach could be further improved by computing segmentation masks at each scale by using the true bounding box locations and expanding the task to include scale dependent object segmentation. The resulting detector would then be analogous to the YoloV5 [79] framework, performing 3 tasks: 1) classification, 2) bounding box regression, 3) scale dependent image segmentation.

Thirdly and lastly, we extend the YoloV4 framework by utilizing RNNs with more sophisticated activation functions like the UR-LSTM or orthogonal projections like the HiPPO-LSTM to fine-tune bounding box trajectories across time. Our experiments show novel RNNs (i.e., UR-LSTM) can lead to desirable performance gains for the video object detection task. UrYolo, which uses a UR-LSTM to fine-tune bounding box trajectories across time, leads to improvements as compared to a simple Yolo baseline and has the highest performance across all models, while also leading to a sparse set of predictions over confidence values. This means that CNNs can benefit from RNNs on spatio-temporal tasks like video object detection. While our SSM formulation Holo, which uses the HiPPO framework shows performance degradations and falls below baseline, this is due to its design. Since HiPPO does not process an entire input vector in parallel and requires the elements in the bounding box vectors to be processed sequentially, the resulting long sequence task becomes a more challenging problem to solve. That is, because HiPPO is designed for long sequence problems, it defines a more challenging problem than UrYolo, which can process entire vectors in parallel, resulting in a short sequence problem. Regardless of that, we conclude that while RNNs, as shown in the UrYolo, can result in an improved object detector that can utilize temporal bounding box information, performance degradation emerges when using the SSM HiPPO framework. That is, using novel RNNs to extract temporal information and use them to fine-tune predictions can be beneficial for the task of video object detection, given that entire vectors can be processed in parallel.

The limitations of HiPPO have however been addressed in S4 [66] and S4ND [81], which may provide an interesting line of future work when examining SSMs on the task of video object detection. S4ND [81] may be particularly interesting, as it does not require inputs to be a flattened vector, meaning that the spatial relationship between tensors is preserved. This makes S4ND similar to 3D convolutions [80] a type of convolutions that are unfolded across time, while maintaining the spatial relationship between frames. Even though, transformers [14] 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. S4 and S4ND on the other hand are SSMs and therefore also have a linear memory increase with respect to input length, making them appealing for tasks

like video object detection, where scalability is crucial. Since the research corpus on SSMs on video object detection is less investigated, exploring SSM-based approaches like S4 and S4ND presents a valuable opportunity that could lead to new insights into optimizing spatio-temporal modeling.

To conclude, our study explored the impact of advanced activation functions and orthogonal projections in RNNs for learning dependencies across (longer) sequences, demonstrated by the UR-LSTM and HiPPO models. While the UrYolo model offered improved performance for spatio-temporal tasks such as video object detection, the HiPPO based Holo model faced challenges due to its design. We also showed that our re-implemention of the YoloV4 object detector led to a slight performance improvement and discussed potential extensions for scale-dependent segmentation. Lastly, we highlighted the promise of more recent SSM-based models like S4 [66] and S4ND [81] for future exploration, suggesting they could overcome the limitations observed with HiPPO in video object detection tasks. These findings emphasize the potential of integrating RNNs and SSMs to enhance object detection for tasks in which spatio-temporal dependencies are relevant.

References

[1] J. Redmon, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016.

[2] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *IEEE transactions on pattern analysis and machine intelligence*, vol. 39, no. 6, pp. 1137–1149, 2016.

[3] H. Wu, Y. Chen, N. Wang, and Z. Zhang, "Sequence level semantics aggregation for video object detection," in *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 9217–9225, 2019.

[4] C. Stauffer and W. E. L. Grimson, "Adaptive background mixture models for real-time tracking," in *Proceedings. 1999 IEEE computer society conference on computer vision and pattern recognition (Cat. No PR00149)*, vol. 2, pp. 246–252, IEEE, 1999.

[5] B. K. Horn and B. G. Schunck, "Determining optical flow," *Artificial intelligence*, vol. 17, no. 1-3, pp. 185–203, 1981.

[6] J. Canny, "A computational approach to edge detection," *IEEE Transactions on pattern analysis and machine intelligence*, no. 6, pp. 679–698, 1986.

[7] I. Sobel, "An isotropic 3× 3 image gradient operater," *Machine vision for three-dimensional scenes*, pp. 376–379, 1990.

[8] D. H. Douglas and T. K. Peucker, "Algorithms for the reduction of the number of points required to represent a digitized line or its caricature," *Cartographica: the international journal for geographic information and geovisualization*, vol. 10, no. 2, pp. 112–122, 1973.

[9] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, pp. 91–110, 2004.

[10] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, vol. 1, pp. 886–893, Ieee, 2005.

[11] B. Babenko, M.-H. Yang, and S. Belongie, "Visual tracking with online multiple instance learning," in *2009 IEEE Conference on computer vision and Pattern Recognition*, pp. 983–990, IEEE, 2009.

[12] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[13] 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.

[14] A. Vaswani, "Attention is all you need," *Advances in Neural Information Processing Systems*, 2017.

[15] A. Dosovitskiy, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[16] K. Cho, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[17] S. Hochreiter, "Long short-term memory," *Neural Computation MIT-Press*, 1997.

[18] Z.-Q. Zhao, P. Zheng, S.-t. Xu, and X. Wu, "Object detection with deep learning: A review," *IEEE transactions on neural networks and learning systems*, vol. 30, no. 11, pp. 3212–3232, 2019.

[19] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 580–587, 2014.

[20] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2117–2125, 2017.

[21] J. Redmon and A. Farhadi, "Yolo9000: better, faster, stronger," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7263–7271, 2017.

[22] A. Farhadi and J. Redmon, "Yolov3: An incremental improvement," in *Computer vision and pattern recognition*, vol. 1804, pp. 1–6, Springer Berlin/Heidelberg, Germany, 2018.

[23] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," *arXiv preprint arXiv:2004.10934*, 2020.

[24] 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," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 7464–7475, 2023.

[25] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14*, pp. 21–37, Springer, 2016.

[26] C.-Y. Fu, W. Liu, A. Ranga, A. Tyagi, and A. C. Berg, "Dssd: Deconvolutional single shot detector," *arXiv preprint arXiv:1701.06659*, 2017.

[27] T.-Y. Ross and G. Dollár, "Focal loss for dense object detection," in *proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2980–2988, 2017.

[28] V. Kalogeiton, V. Ferrari, and C. Schmid, "Analysing domain shift factors between videos and images for object detection," *IEEE transactions on pattern analysis and machine intelligence*, vol. 38, no. 11, pp. 2327–2334, 2016.

[29] X. Zhu, Y. Wang, J. Dai, L. Yuan, and Y. Wei, "Flow-guided feature aggregation for video object detection," in *Proceedings of the IEEE international conference on computer vision*, pp. 408–417, 2017.

[30] S. Wang, Y. Zhou, J. Yan, and Z. Deng, "Fully motion-aware network for video object detection," in *Proceedings of the European conference on computer vision (ECCV)*, pp. 542–557, 2018.

[31] G. Bertasius, L. Torresani, and J. Shi, "Object detection in video with spatiotemporal sampling networks," in *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 331–346, 2018.

[32] F. Xiao and Y. J. Lee, "Spatial-temporal memory networks for video object detection," *arXiv preprint arXiv:1712.06317*, 2017.

[33] 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*, pp. 3909–3918, 2019.

[34] M. Shvets, W. Liu, and A. C. Berg, "Leveraging long-range temporal relationships between proposals for video object detection," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 9756–9764, 2019.

[35] Y. Chen, Y. Cao, H. Hu, and L. Wang, "Memory enhanced global-local aggregation for video object detection," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10337–10346, 2020.

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

[37] K. Kang, H. Li, J. Yan, X. Zeng, B. Yang, T. Xiao, C. Zhang, Z. Wang, R. Wang, X. Wang, *et al.*, "T-cnn: Tubelets with convolutional neural networks for object detection from videos," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 28, no. 10, pp. 2896–2907, 2017.

[38] K. Kang, W. Ouyang, H. Li, and X. Wang, "Object detection from video tubelets with convolutional neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 817–825, 2016.

[39] K. Kang, H. Li, T. Xiao, W. Ouyang, J. Yan, X. Liu, and X. Wang, "Object detection in videos with tubelet proposal networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 727–735, 2017.

[40] 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," *arXiv preprint arXiv:1602.08465*, 2016.

[41] C. Feichtenhofer, A. Pinz, and A. Zisserman, "Detect to track and track to detect," in *Proceedings of the IEEE international conference on computer vision*, pp. 3038–3046, 2017.

[42] X. Zhu, J. Dai, X. Zhu, Y. Wei, and L. Yuan, "Towards high performance video object detection for mobiles," *arXiv preprint arXiv:1804.05830*, 2018.

[43] T. Gong, K. Chen, X. Wang, Q. Chu, F. Zhu, D. Lin, N. Yu, and H. Feng, "Temporal roi align for video object recognition," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, pp. 1442–1450, 2021.

[44] Y. Shi, N. Wang, and X. Guo, "Yolov: Making still image object detectors great at video object detection," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 37, pp. 2254–2262, 2023.

[45] S. Tripathi, Z. C. Lipton, S. Belongie, and T. Nguyen, "Context matters: Refining object detection in video with recurrent neural networks," *arXiv preprint arXiv:1607.04648*, 2016.

[46] Y. Lu, C. Lu, and C.-K. Tang, "Online video object detection using association lstm," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2344–2352, 2017.

[47] 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*, pp. 6678–6687, 2019.

[48] J. Deng, Y. Pan, T. Yao, W. Zhou, H. Li, and T. Mei, "Relation distillation networks for video object detection," in *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 7023–7032, 2019.

[49] A. Mhalla, T. Chateau, and N. E. B. 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.

[50] Y. Chen, Y. Cao, H. Hu, and L. Wang, "Memory enhanced global-local aggregation for video object detection," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10337–10346, 2020.

[51] 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.

[52] A. Gu, C. Gulcehre, T. Paine, M. Hoffman, and R. Pascanu, "Improving the gating mechanism of recurrent neural networks," in *International conference on machine learning*, pp. 3800–3809, PMLR, 2020.

[53] A. Gu, T. Dao, S. Ermon, A. Rudra, and C. Ré, "Hippo: Recurrent memory with optimal polynomial projections," *Advances in neural information processing systems*, vol. 33, pp. 1474–1487, 2020.

[54] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*, pp. 740–755, Springer, 2014.

[55] 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.

[56] D. Gorden, A. Farhadi, and D. Fox, "Re3: Real-time recurrent regression networks for object tracking," *IEEE Robot. Autom. Mag.*, vol. 3, no. 2, pp. 788–795, 2018.

[57] 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, IEEE, 2019.

[58] 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, p. 9700, 2022.

[59] 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, IEEE, 2019.

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

[61] R. Pascanu, "On the difficulty of training recurrent neural networks," *arXiv preprint arXiv:1211.5063*, 2013.

[62] Y. LeCun and N. EDU, "Orthogonal rnns and long-memory tasks,"

[63] J. S. Bayer, *Learning sequence representations*. PhD thesis, Technische Universität München, 2015.

[64] C. Tallec and Y. Ollivier, "Can recurrent neural networks warp time?," *arXiv preprint arXiv:1804.11188*, 2018.

[65] R. Jozefowicz, W. Zaremba, and I. Sutskever, "An empirical exploration of recurrent network architectures," in *International conference on machine learning*, pp. 2342–2350, PMLR, 2015.

[66] A. Gu, K. Goel, and C. Ré, "Efficiently modeling long sequences with structured state spaces," *arXiv preprint arXiv:2111.00396*, 2021.

[67] A. Gu and T. Dao, "Mamba: Linear-time sequence modeling with selective state spaces," *arXiv preprint arXiv:2312.00752*, 2023.

[68] Y. Liu, Y. Tian, Y. Zhao, H. Yu, L. Xie, Y. Wang, Q. Ye, and Y. Liu, "Vmamba: Visual state space model 2024," *arXiv preprint arXiv:2401.10166*, 2024.

[69] K. Ogata, "Modern control engineering," 2020.

[70] A. Simpkins, "System identification: Theory for the user, (ljung, l.; 1999)[on the shelf]," *IEEE Robotics & Automation Magazine*, vol. 19, no. 2, pp. 95–96, 2012.

[71] K. J. Åström and R. Murray, *Feedback systems: an introduction for scientists and engineers*. Princeton university press, 2021.

[72] A. Voelker, I. Kajić, and C. Eliasmith, "Legendre memory units: Continuous-time representation in recurrent neural networks," *Advances in neural information processing systems*, vol. 32, 2019.

[73] R. A. DeCarlo, *Linear systems: A state variable approach with numerical implementation*. Prentice-Hall, Inc., 1989.

[74] A. Tustin, "A method of analysing the behaviour of linear systems in terms of time series," *Journal of the Institution of Electrical Engineers-Part IIA: Automatic Regulators and Servo Mechanisms*, vol. 94, no. 1, pp. 130–142, 1947.

[75] A. M. Lamb, A. G. ALIAS PARTH GOYAL, Y. Zhang, S. Zhang, A. C. Courville, and Y. Bengio, "Professor forcing: A new algorithm for training recurrent networks," *Advances in neural information processing systems*, vol. 29, 2016.

[76] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *International conference on machine learning*, pp. 6105–6114, PMLR, 2019.

[77] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," *IEEE transactions on pattern analysis and machine intelligence*, vol. 37, no. 9, pp. 1904–1916, 2015.

[78] S. Liu, L. Qi, H. Qin, J. Shi, and J. Jia, "Path aggregation network for instance segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 8759–8768, 2018.

[79] G. Jocher, A. Stoken, J. Borovec, L. Changyu, A. Hogan, L. Diaconu, F. Ingham, J. Poznanski, J. Fang, L. Yu, *et al.*, "ultralytics/yolov5: v3. 1-bug fixes and performance improvements," *Zenodo*, 2020.

[80] D. Tran, L. D. Bourdev, R. Fergus, L. Torresani, and M. Paluri, "C3d: generic features for video analysis," *CoRR, abs/1412.0767*, vol. 2, no. 7, p. 8, 2014.

[81] E. Nguyen, K. Goel, A. Gu, G. Downs, P. Shah, T. Dao, S. Baccus, and C. Ré, "S4nd: Modeling images and videos as multidimensional signals with state spaces," *Advances in neural information processing systems*, vol. 35, pp. 2846–2861, 2022.

[82] G. B. Arfken, H. J. Weber, and F. E. Harris, *Mathematical methods for physicists: a comprehensive guide*. Academic press, 2011.

I Appendix

I.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:*

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

Proof. Using Rodrigues' formulation [82] of the Legendre Polynomial $P_n x$:

$$P_n(x) = \frac{1}{2^n n!}\frac{d^n}{dx^n}\left[\left(x^2-1\right)^n\right], \tag{6.1}$$

and considering an integral over the interval $[-1, 1]$ of a function f:

$$\int_{-1}^{1} f(x)P_n(x)dx, \tag{6.2}$$

then substituting $P_n(x)$ with Rodrigues' formulation from Eq. 6.1 into Eq. 6.2 we have:

$$\begin{aligned}&\int_{-1}^{1} f(x)P_n(x)dx\\ =&\int_{-1}^{1} f(x)\frac{1}{2^n n!}\frac{d^n}{dx^n}\left[\left(x^2-1\right)^n\right]dx\end{aligned} \tag{6.3}$$

Performing integration by parts n times using:

$$\int udv = uv - \int vdu, \tag{6.4}$$

and defining u and dv as:

$$u = f(x) \tag{6.5}$$

$$dv = \frac{1}{2^n n!}\frac{d^n}{dx^n}\left[\left(x^2-1\right)^n\right]dx, \tag{6.6}$$

we define $g(x)$ so that:

$$g(x) = \left(x^2-1\right)^n. \tag{6.7}$$

Plugging $g(x)$ from Eq. (6.7) into dv in Eq. (6.6):

$$dv = \frac{1}{2^n n!}\frac{d^n g(x)}{dx^n}dx, \tag{6.8}$$

and integrating dv w.r.t. x:

$$v = \int \frac{1}{2^n n!}\frac{d^n g(x)}{dx^n}dx, \tag{6.9}$$

where the anti-derivative of the derivative is:

$$v = \frac{1}{2^n n!}g(x) = \frac{1}{2^n n!}\left(x^2-1\right)^n \tag{6.10}$$

Finally, applying integration by parts:

$$\int_{-1}^{1} f(x)\frac{d^n}{dx^n}\left[\left(x^2-1\right)^n\right]dx \tag{6.11}$$

$$=\left[f(x)\frac{1}{2^n n!}\left(x^2-1\right)^n\right]_{-1}^{1}-\int_{-1}^{1}\frac{d}{dx}f(x), \tag{6.12}$$

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

$$-\int_{-1}^{1}\frac{d}{dx}f(x)\cdot\frac{1}{2^n n!}\left(x^2-1\right)^n dx. \tag{6.13}$$

Integrating n times, we get:

$$(-1)^n\int_{-1}^{1}f^{(n)}(x)\cdot\frac{1}{2^n n!}\left(x^2-1\right)^n dx, \tag{6.14}$$

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

$$\int_{-1}^{1}f(x)P_n(x)dx=\frac{(-1)^n}{2^n n!}\int_{-1}^{1}f^{(n)}(x)\left(x^2-1\right)^n dx. \tag{6.15}$$

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:

$$f^{(n)}(x)=\frac{d^n}{dx^n}[P_m(x)]=0, \tag{6.16}$$

and this results follows.

■

I.ii Eigenvalues of Orthogonal Matrices

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

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

$$Av=\lambda v \tag{6.17}$$

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

$$\|Av\|=\|v\| \tag{6.18}$$

Consider the norm of Av:

$$\|Av\|^2=(Av)^T(Av)=(\lambda v)^T(\lambda v)=\lambda^2 v^Tv=\lambda^2\|v\|^2 \tag{6.19}$$

Since A is orthogonal, we also have:

$$\|Av\|^2=\|v\|^2 \tag{6.20}$$

From the above two equations, we equate the norms:

$$\lambda^2\|v\|^2=\|v\|^2 \tag{6.21}$$

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

$$\lambda^2 = 1 \tag{6.22}$$

Taking the square root of both sides, we find:

$$|\lambda| = 1 \tag{6.23}$$

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

■

I.iii Code Repository and Demonstrations

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