Learning Structure from Motion
DRAFT

Masterarbeit

der Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

vorgelegt von

Adrian Wälchli

2017

Leiter der Arbeit:
Prof. Dr. Paolo Favaro
Institut für Informatik und angewandte Mathematik

Abstract

Abstract

Acknowledgements

ack

Contents

Chapter 1

Prior Work

Add text that introduces the chapter
Find good structure/order to present the different works
Better point out differences and similarities
Typeset vectors/matrices with custom commands

Possible definition of structure from motion by **?**:

"The structure from motion (SfM) problem in computer vision is the problem of recovering the three-dimensional (3D) structure of a stationary scene from a set of projective measurements, represented as a collection of two-dimensional (2D) images, via estimation of motion of the cameras corresponding to these images."

One can define the three basic steps of SfM as:

1. Feature detection, extraction and matching
2. Camera motion estimation
3. Recovery of 3D structure

This chapter presents the prior work that focuses on one or multiple of these steps.

1.1 Feature-based methods SURF, ORB, SIFT

1.2 The Eight Point Algorithm

The eight point algorithm, as introduced by **?**, makes use of the epipolar geometry between a pair of views in order to compute relative translation and orientation of the cameras. The relationship between the two camera views is described by the so-called *essential matrix*. It is defined as

$$E = [t]_\times R, \tag{1.1}$$

where $[t]_\times$ is the matrix that computes the cross-product between t and an arbitrary vector. It describes the relationship between corresponding points in the image planes of the two cameras. When R and t are unknown, it is possible to compute the essential matrix using known corresponding points. The epipolar constraint is formulated as

$$\hat{x}_1^\top E \hat{x}_0 = 0, \tag{1.2}$$

where $\hat{x}_0$ and $\hat{x}_1$ are the corresponding points in image one and two respectively. The unknown elements of the essential matrix can be determined using this system of equations. It requires eight equations for eight elements of the matrix and one element has to be fixed because any scaling of the true matrix satisfies the epipolar constraint. After estimating E, the rotation and translation can be recovered from it up to some ambiguities.

In practice, there are some problems that need to be addressed. First, the corresponding points are usually not known and must be found using a feature detection and matching technique (SfM step 1). This then can lead to outliers which are wrong correspondences. In this case, RANSAC is applied to select the best eight points among many possible choices. Second, the selected points are usually not exact and contain noise that lead to a rank three estimate of E instead of rank two. The singular value decomposition is used to discard the component with the smallest singular value. **numerical issues, 5 point algorithm** Third, in some cases the calibration matrices K_0 and K_1 are unknown. This means that the normalized coordinates are $\hat{x}_j = K_j^{-1} x_j$ are no longer available and only the image coordinates x_j are known. This results in the so called *fundamental matrix* F in the epipolar constraint:

$$\hat{x}_1^\top E \hat{x}_0 = x_1^\top K_1^{-\top} E K_0^{-1} x_0 = x_1^\top F x_0 = 0. \tag{1.3}$$

explain what F is used for

1.3 Factorization Methods

The factorization method can be used to recover camera motion and 3D shape simultaneously (step 2 and 3). A first method was introduced by **?** that works with an orthographic camera model. It uses the so called *measurement matrix* $W \in \mathbb{R}^{2n \times m}$ which is defined as

$$W = \begin{bmatrix} x_{11} & \cdots & x_{1m} \\ \vdots & \ddots & \vdots \\ x_{n1} & \cdots & x_{nm} \\ y_{11} & \cdots & y_{1m} \\ \vdots & \ddots & \vdots \\ y_{n1} & \cdots & y_{nm} \end{bmatrix}. \tag{1.4}$$

It contains the 2D coordinates of the m orthographically projected 3D points for each of the n cameras. Under the assumption that the origin of global coordinate system is located at the centroid of the point cloud, the matrix W can be factorized into

$$W = MS, \tag{1.5}$$

where $M \in \mathbb{R}^{2n \times 3}$ contains the 2×3 projection matrices of each camera and $S \in \mathbb{R}^{3 \times m}$ are the points of the observed 3D shape. As described by **?**, this factorization can be achieved using the singular value decomposition (SVD) as

$W = U\Sigma V^\top$ and deriving M and S from U, V and Σ. **Add more details? Less details?**

This factorization method has some disadvantages. First, it can not be applied to the commonly used perspective projection model, although it is approximated by the orthographic model for distant objects. The works of **?** and **?** have extended the factorization method for perspective projection using an iterative process. Second, in order for the factorization method to work, all points of the shape must be visible in every frame, i.e. to obtain the measurement matrix one needs to find the correspondences across all frames.

1.4 Bundle Adjustment

Another method to solve multi-view SfM is the method of *bundle adjustment*. It is the most general technique for simultaneously recovering 3D shape and camera pose, but it is also computationally expensive. The bundle adjustment technique aims to minimize the re-projection error of the unknown 3D points and camera matrices. The re-projection error is formulated as the euclidean distance between the known image coordinates and the projection of the unknown points using the unknown camera matrices.

More formally, let $\{C_j\}_{j=1}^m$ be the (unknown) camera matrices that encode location, rotation and intrinsic parameters, let $\{p_i\}_{i=1}^n$ denote the (unknown) 3D points and let x_{ij} be the (known) observed 2D coordinates of the point p_i in camera j. Then, the objective of bundle adjustment is defined as

$$\min_{\{p_i\},\{C_j\}} \quad \sum_{i=1}^{n}\sum_{j=1}^{m} v(i,j)\,\|P(C_j,p_i) - x_{ij}\|_2\,, \tag{1.6}$$

where $v(i,j)$ denotes the visibility of point i in camera j and P is the projection operation with homogeneous division. This optimization problem is non-convex and as explained by **?** naïve optimization algorithms achieve only a poor local minimum. One approach to find a better local minimum is to initialize using an other SfM algorithm such as the factorization method.

1.5 2015 - PoseNet

The PoseNet, as proposed by **?**, is a CNN that performs regression for the location and orientation of the camera given a single image as input. It outputs a 7D vector that describes the pose $p = [x, q]^\top$ as the camera location x (3D) and orientation q (4D quaternion). They simultaneously learn location and orientation with the euclidean loss

$$\mathcal{L}(I) = \|\hat{x} - x\|_2 + \beta \left\|\hat{q} - \frac{q}{\|q\|}\right\|_2\,, \tag{1.7}$$

where β is a parameter to balance the expected error of pose and orientation. The architecture is a modified *GoogleNet* with 23 layers which was trained on a classification task. They replace the last layers to perform regression instead of classification.

Because CNNs require a large amount of training data, the authors apply transfer learning by pretraining the network on a different task with large datasets such as *ImageNet* and *Places*. Pose regression is performed using the pre-trained network and their own dataset called *Cambridge Landmarks* with five scenes of camera motion in large scale outdoor areas. For this limited dataset, ground truth pose is available. A challenge that comes with this dataset is the clutter in form of moving pedestrians and cars. Also, due to the long trajectories, weather and lighting conditions change a lot.

In their experiments they find that the system is robust to large spatial distance between camera samples. Also, from the visualization of the features they observe that the network does not only produce high response outputs for high level features, but is also sensitive to large textureless regions where SIFT-based approaches typically fail.

The paper demonstrates that transfer-learning can be used for pose estimation when a large labeled dataset for training is not available and that the pre-trained network can learn pose information despite being forced to produce pose-invariant outputs. **connection to other work?**

1.6 04/2016 - 3d-r2n2: A unified approach for single and multi-view 3d object reconstruction

?

1.7 07/2016 - Unsupervised CNN for Single View Depth Estimation

The work of **?** implements a autoencoder on stereo pair images to predict depth from a single image. Their architecture consist of the following parts.

- **Encoder**
  Takes a single image as input. At training time, this is the left image of a stereo pair. The output is the predicted disparity map (scaled inverse depth).

- **Decoder**
  The decoder is only used at training time. It takes two inputs: The predicted disparities from the encoder and the right image of the stereo pair. The right image is warped using the displacements and compared to the encoder input (left image) using the color constancy error (photometric error).

Since the stereo pair is rectified, the disparity is a displacement along the scanline of the images. Thus the decoder implements a simple geometric transformation that does not need to be learned. At test time, only the encoder network with a single image as input is used.

As noted by the author, there are standard stereo algorithms that produce disparity maps. However, these methods can not deal with distortions such as lens flare, motion blur, shadows, etc. The idea is that a neural network could learn to deal with such problems that occur in natural images.

The dataset they use is KITTI **?** which they augment by random crops, color channel scaling, and flipping the images.

1.8 04/2017 - SfM-Net

? implement a deep learning approach to structure from motion. Their architecture consists of two subnetworks:

- **Structure**
  Learns per-frame depth. The input is a single frame. The output of the CNN is a cloud of 3D points, one for each pixel value in the input image.
- **Motion**
  The input is a pair of frames. The CNN computes a set of K segmentation masks for moving objects. Using these masks, each pixel is assigned to an object specific transformation given by a rotation and a translation. In addition, camera rotation and translation are computed using the features of the inner layers.

Given a pair of images, the forward operation of the network works as follows:

1. The structure network computes the point cloud for the first frame.
2. The motion network computes the object transformations as well as the camera transformation using both frames.
3. The point cloud is transformed using the learned object transformations and masks.
4. The transformed 3D points are re-projected to 2D using the learned camera transformation between the two frames.

The transformed point cloud corresponds to the depth of the second frame. Optical flow can be computed directly from the re-projected points.

The authors of the paper propose various modes of supervision to evaluate the architecture and to handle ambiguities in the reconstruction due to the ill-posed problem:

- **Self-supervision**
  No ground truth is given. The loss is defined by the brightness constancy constraint of the second frame warped to the first frame using the predicted optical flow. For this mode, they use the KITTI 2012/15 datasets.
- **Depth**
  Ground truth is given in the form of depth for each pixel. This can be acquired for example by a Kinect sensor. They use the RGB-D SLAM dataset for ground truth depth. This helps to improve camera motion estimation.
- **Camera motion**
  Camera motion is given as ground truth in form of a rotation and translation matrix. The relative transformation between predicted and ground truth transformation is

- **Optical flow and object motion**
  They use this type of supervision with the MoSeg dataset which contains ground truth segmentation for each frame. This dataset contains more non-rigid body transformation. They evaluate the quality of the object motion mask by intersection over union (IoU).

1.9 04/2017 - Unsupervised Learning of Depth and Ego-Motion from Video

This work from **?** is concurrent to the SfM-Net by **?**. Both works share a similar concept. However, **?** focus on unsupervised training and do not explicitly model scene dynamics. Similar to SfM-Net, their architecture consists of two jointly trained networks, one for learning depth from a single image (Depth-CNN), and another for learning camera motion from two frames (Pose-CNN).

The key difference to SfM-Net is that they train with an image sequence $I_1, \ldots, I_N$ with one image being the target image I_t. The task for the motion network is to synthesize the target image from two nearby views I_{t-1} and I_{t+1}.

- **Depth-CNN**
  Takes the target image I_t as input and predicts a per-pixel depth map $\hat{D}_t$.
- **Pose-CNN**
  Uses two source images I_{t-1} and I_{t+1} as input and predicts relative camera poses $\hat{T}_{t \to t-1}$ and $\hat{T}_{t \to t+1}$. These transformations are then used to synthesize the target image.

Using both the estimated depth and pose matrix as well as the intrinsic matrix of the camera, the coordinate frame of the source images is warped to the coordinate frame of the target image. To transfer the color information from one frame to the other, bilinear interpolation is used. The general term for this is *differentiable image warping*.

When working with realistic data, it is important to make the model robust to non-static objects in the scenes, occlusions and non-Lambertian surfaces. To do this, the network is forced to additionally learn a per-pixel mask for each target-source pair. They add a regularization term to force the mask to have non-zero values.

1.10 04/2017 - DeMoN: Depth and Motion Network for Learning Monocular Stereo

? propose a network architecture to learn depth, motion as well as optical flow and surface normals from two images. The network architecture consists of three subnetworks in sequence:

- **Bootstrap network**
  Consists of a pair of encoder-decoder networks. The first encoder-decoder predicts optical flow and its confidence. The second encoder-decoder takes the previously computed optical flow and confidence, as well as the original image pair and a warp of the second image using the estimated flow.

In addition, there are three fully connected layers predicting motion. By taking the image pair as input, the network can make use of motion parallax.

- **Iterative network**
  The architecture is identical to the bootstrap network, but takes additional inputs. It is applied iteratively with the idea to improve previous estimates of depth, normals and motion. The network is trained using four iterations.
- **Refinement network**
  Up-samples the low resolution predictions to the full image resolution.

Motion is parameterized using an axis-angle representation and depth is estimated by its inverse value $1/z$ in order to represent points at infinity. For surface normals, motion and optical flow they use the euclidean loss and for depth L1. In addition to the point wise loss for depth, they define a scale invariant gradient loss that penalizes relative depth errors between neighboring pixels.

They evaluate the performance of the network on various datasets: SUN3D, RGB-D SLAM, MVS, Scenes11, and NYUv2. The results show that estimating normals and flow in addition to depth and motion slightly improves the performance of the system. Compared to works for depth from a single image, their results demonstrate that the network learns to use motion parallax from the stereo images to generalize better to scenes that heavily differ from the examples seen during training.

1.11 05/2017 - Relative Camera Pose Estimation Using Convolutional Neural Networks

? propose a CNN architecture to estimate the relative pose between two cameras. Similar to PoseNet **?**, they estimate the 7D vector of pose by a quaternion and a translation vector. Their network is composed of two main parts.

- **Representation part**
  The inputs are the two camera images. This part has two identical branches with shared weights of five convolutional layers with ReLUs and spatial pyramid pooling at the end. Each branch processes one of the input images.
- **Regression part**
  Regression is performed on the output of the representation part using two fully-connected layers to estimate the relative pose.

The weights for the Siamese network are initialized from a pre-trained AlexNet on the *ImageNet* and *Places* datasets. In order to overcome the limitation in the input size due to the fully-connected layers, **?** apply spatial pyramid pooling (SPP) in the representation part. This allows for a larger size of input images for the network to be able to extract more structural information that helps reconstructing the pose.

The dataset used for training is a crowd-sourced image collection with ground truth pose from a structure from motion pipeline based on local feature matching. For testing and comparison with other works they use the DTU dataset where the exact pose is obtained from a robotic arm that moves the camera. **incomplete summary**

Chapter 2

Introduction

2.1 Relative Camera Pose

Equations to transform pose in world coordinates to relative pose with respect to the coordinate frame of the first camera.

2.2 Rotation Metrics

?

2.3 Two-Frame Structure from Motion

2.4 Multiview Structure from Motion

2.5 Camera Pose and Transformations

The camera pose, also known as the *extrinsic parameters*, is a combination of position and orientation. Together, they define a coordinate transformation from the camera coordinate frame to the world coordinate frame. Such a transformation is commonly represented by a translation vector $\mathbf{t} \in \mathbb{R}^3$ and a rotation matrix $\mathbf{R} \in SO(3)$. Together, they describe a rigid transformation which can also be written as a 4×4 matrix of the form

$$\mathbf{T} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ 0 & 1 \end{bmatrix} \in SE(3). \tag{2.1}$$

Insert a figure showing: world coordinate frame, camera coordinate frame, transformation matrix

In Structure from Motion, there is not only one camera, but many cameras with different transformations. This is also the case in this work, but the sequence of transformations is ordered according to the path of the camera motion. Specifically, we deal with a sequence of rotations $\mathbf{R}_1, \ldots, \mathbf{R}_n$ and translations $\mathbf{t}_1, \ldots, \mathbf{t}_n$. Depending on how the poses are obtained, they are all relative to a common (world) coordinate frame. Sometimes it is useful to have

	Euler angles	Matrix	Axis-angle	Unit quaternion
Typical use case	Robotics, Avionics	Computer Graphics, Physics	Intermediate representation	Computer Graphics, Physics
Constraint	None	Orthonormality	Unit norm axis	4D unit sphere
Disadvantage	Gimbal lock	Numerically unstable		Less intuitive
Interpolation	hard	hard	hard	easy
Identity rotation	ambiguous	unique	ambiguous	unique

Table 2.1: text

the coordinate frame of the first camera transform be the common frame for all transformations. This can be achieved by applying the transformations

$$\begin{aligned}\mathbf{R}_i' &= \mathbf{R}_1^\top \mathbf{R}_i \\ \mathbf{t}_i' &= \mathbf{R}_1^\top (\mathbf{t}_i - \mathbf{t}_1)\end{aligned} \tag{2.2}$$

to get the new relative rotation $\mathbf{R}_i'$ and relative translation $\mathbf{t}_i'$. Note that for $i = 1$ we obtain $\mathbf{R}_1' = \mathbf{I}$ and $\mathbf{t}_1' = \mathbf{0}$.

——————— **make good transition**

So far, we have seen rotations represented as matrices. However, it is redundant to describe rotations with nine numbers when in fact there are only three degrees of freedom for a three-dimensional rotation, i.e. two degrees for the orientation of the axis and one for the angle of rotation around the axis. It is also not straightforward to interpolate rotations in $SO(3)$, or to find the best approximation for a matrix that lies outside of $SO(3)$.**citation needed**

Below we discuss and compare three other popular representations used to describe rotations. A brief overview is also shown in table **??**.

Euler Angles

Axis-Angle

Quaternions Quaternions are four-dimensional numbers. They are an extension of the complex numbers[1]. Formally, a quaternion is defined as $\mathbf{q} = w + \mathbf{i}x + \mathbf{j}y + \mathbf{k}z$ where $w, x, y, z \in \mathbb{R}$ and $\mathbf{i}, \mathbf{j}$ and $\mathbf{k}$ are the basis elements of the quaternion space. The quaternion can also be written as a tuple $\mathbf{q} = (w, x, y, z) \in \mathbb{R}^4$ by using the notation

$$\mathbf{1} = \begin{bmatrix}1\\0\\0\\0\end{bmatrix}, \quad \mathbf{i} = \begin{bmatrix}0\\1\\0\\0\end{bmatrix}, \quad \mathbf{j} = \begin{bmatrix}0\\0\\1\\0\end{bmatrix}, \quad \mathbf{k} = \begin{bmatrix}0\\0\\0\\1\end{bmatrix}. \tag{2.3}$$

But quaternions are not just tuples of numbers. They are equipped with a separate addition and multiplication. Addition is simply carried out element-wise as for vectors, whereas multiplication makes use of the distributive rule

[1]A more appropriate name is *compound numbers*.**disputable**

and the property

$$\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = \mathbf{ijk} = -1. \tag{2.4}$$

Furthermore, the norm of a quaternion is defined as

$$\|\mathbf{q}\| = \sqrt{w^2 + x^2 + y^2 + z^2}. \tag{2.5}$$

In this thesis, we are particularly interested in the subset of quaternions called the *unit quaternions*. These are all the quaternions with unit norm, i.e. $\|\mathbf{q}\| = 1$. The quaternions of this form describe 3D rotations and can be parameterized by

$$\mathbf{q} = \cos\left(\frac{\theta}{2}\right) + \sin\left(\frac{\theta}{2}\right)(a_1\mathbf{i} + a_2\mathbf{j} + a_3\mathbf{k}), \tag{2.6}$$

where $\theta \in \mathbb{R}$ is the angle of rotation around a unit-length axis $\mathbf{a} = (a_1, a_2, a_3) \in \mathbb{R}^3$. **explain that quaternions are not unique representation of rotations, and that negative axis-angle is represented by the same quaternion**

2.6 Recurrent Neural Networks

Need references to early works on RNNs So far, we have looked at the feed-forward network, which is function that maps an input $\mathbf{x}$ to an output $\mathbf{y} = f(\mathbf{x})$. It will always compute the same output for the same input, independently of the input-output pairs it has seen before. Now consider a sequence $\mathbf{x}_1, \dots, \mathbf{x}_T$ of inputs (e.g. a sentence). How can we learn to predict an attribute that depends on the whole sequence (e.g. the sentiment of the sentence)? One way is to concatenate the sequence to one vector and use the feed-forward network. However, this will limit the sequence length to a fixed size and potentially requires more training data. **need reference for this statement**. How do we model predictions for a variable sequence length? And what if we want also the output to be a sequence? What if the output sequence needs to have a different length than the input sequence, such as in machine translation? **citation needed**

This is where the recurrent neural network (RNN) comes into play. It is a function that maps each vector $\mathbf{x}_t \in \mathbb{R}^n$ from the sequence to a hidden state

$$\mathbf{h}_t = R(\mathbf{x}_t, \mathbf{h}_{t-1}), \qquad t = 1, \dots, T \tag{2.7}$$

with the hidden state $\mathbf{h}_{t-1} \in \mathbb{R}^d$ carried over from the previous prediction as a second input, where d is called the *hidden size* of the RNN. The hidden state represents the accumulated knowledge from all previous inputs. This way, the output at each time step t depends on the inputs from $\mathbf{x}_1$ to $\mathbf{x}_{t-1}$. Figure 2.1 shows the RNN as a looping component. When unfolded, one can see the information flow from one state to the next. Note that it is possible to feed arbitrary sizes of sequences to the RNN.

The simplest implementation of an RNN uses an affine transformation followed by a non-linear function:

$$\mathbf{h}_t = \tanh\left(\mathbf{W}\begin{bmatrix}\mathbf{x}_t \\ \mathbf{h}_{t-1}\end{bmatrix} + \mathbf{b}\right) \tag{2.8}$$

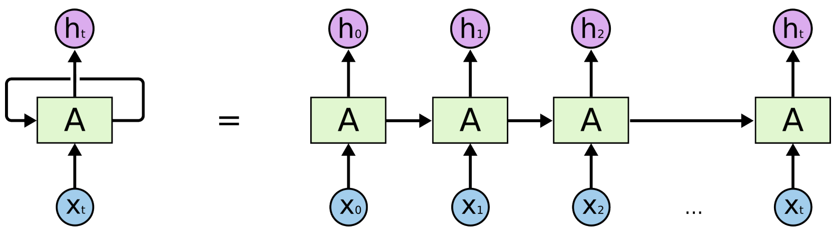

Figure 2.1: Unfolding the RNN. Left: The RNN as a cyclic graph. Right: Unfolded RNN for a number of timesteps. Image courtesy Christopher Olah (colah.github.io)

The weight matrix $\mathbf{W}$ has dimensions $d \times (n+d)$. Sometimes it is desirable to have an output that has a different size than the hidden state (e.g. for classification). In this case one can apply a second affine layer

$$\mathbf{y}_t = \mathbf{V}\mathbf{h}_t + \mathbf{c} \tag{2.9}$$

to obtain the prediction $\mathbf{y}_t$. Otherwise we define the output as $\mathbf{y}_t = \mathbf{h}_t$.

Training an RNN is not much different from a feed-forward network. The weights $\mathbf{W}, \mathbf{V}, \mathbf{b}$ and $\mathbf{c}$ are updated in the same fashion as for feed-forward networks using gradient descent on the loss function. Say we define a criterion $L_t(\mathbf{y}_t, \hat{\mathbf{y}}_t)$ for the loss between prediction and ground-truth at each time step t. Then, the total loss for the sequence is simply

$$L(\{\mathbf{y}_1, \dots \mathbf{y}_T\}, \{\hat{\mathbf{y}}_1, \dots \hat{\mathbf{y}}_T\}) = \sum_{t=1}^{T} L_t(\mathbf{y}_t, \hat{\mathbf{y}}_t). \tag{2.10}$$

The gradient computation is almost the same as for feed-forward networks and the back-propagation algorithm can be applied by unfolding the RNN as shown in figure 2.1. However, there is a problem with the gradient computation in the above version of the RNN (**?**, **?**). For long sequences, the gradient norm can become very small and this hurts the training. The RNN is not able to learn long-term dependencies when the gradient vanishes. To see this, let's write down the full equation for the gradient:

$$\frac{\partial L}{\partial \mathbf{w}} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial \mathbf{w}} = \sum_{t=1}^{T} \sum_{k=0}^{t} \frac{\partial L_t}{\partial \mathbf{y}_t} \frac{\partial \mathbf{y}_t}{\partial \mathbf{h}_t} \left(\prod_{j=k+1}^{t} \frac{\partial \mathbf{h}_j}{\partial \mathbf{h}_{j-1}} \right) \frac{\partial \mathbf{h}_k}{\partial \mathbf{w}} \tag{2.11}$$

In the above equation, we have the product term that causes the problem when the sequence is very long. Note that the partial derivatives are Jacobian matrices when we take the derivatives of a vector-valued function with respect to a vector. Due to many multiplications, the gradient values drop exponentially. **Need a better explanation why values are ¡1 and drop exponentially** A more complete analysis is provided in **?**.

2.7 The Long Short-Term Memory

The Long Short-Term Memory (LSTM) is a version of the RNN that was designed to overcome the vanishing gradient problem described above, and it is

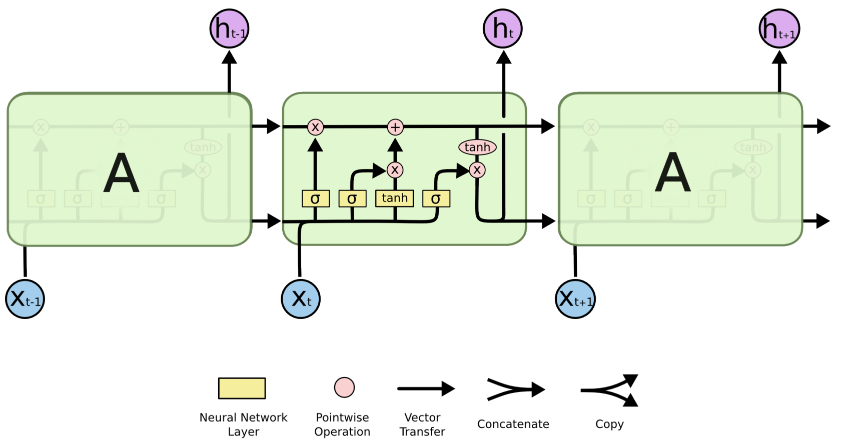

Figure 2.2: Unfolded LSTM cell. Image courtesy Christopher Olah (colah.github.io)

also used in this thesis. As shown in figure 2.2, it introduces four gates (in yellow) and an additional cell state variable that is carried over from one time step to the next. The recurrence can be formulated as a function

$$(\mathbf{h}_t, \mathbf{c}_t) = R(\mathbf{x}_t, \mathbf{h}_{t-1}, \mathbf{c}_{t-1}), \qquad t = 1, \ldots, T. \tag{2.12}$$

More precisely, the hidden state and cell state are computed as follows:

$$\begin{aligned} \begin{bmatrix} \mathbf{i} \\ \mathbf{f} \\ \mathbf{o} \\ \mathbf{g} \end{bmatrix} &= \begin{bmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{bmatrix} \left(\mathbf{W} \begin{bmatrix} \mathbf{x}_t \\ \mathbf{h}_{t-1} \end{bmatrix} + \begin{bmatrix} \mathbf{b}_i \\ \mathbf{b}_f \\ \mathbf{b}_o \\ \mathbf{b}_g \end{bmatrix} \right) \\ \mathbf{c}_t &= \mathbf{i} \odot \mathbf{g} + \mathbf{f} \odot \mathbf{c}_{t-1} \\ \mathbf{h}_t &= \mathbf{o} \odot \tanh(\mathbf{c}_t) \end{aligned} \tag{2.13}$$

The matrix $\mathbf{W} \in \mathbb{R}^{4d \times (n+d)}$ contains the weights for the input- and hidden state transformations before each gate. It can be broken down into four parts:

$$\mathbf{W} = \begin{bmatrix} \mathbf{W}_i \\ \mathbf{W}_f \\ \mathbf{W}_o \\ \mathbf{W}_g \end{bmatrix} \tag{2.14}$$

We can see that there are three additional gates that control information flow with a sigmoid activation. If the sigmoid output is one, all information is let through, otherwise no information passes the gate.

The first is the input gate $\mathbf{i}$. It controls how much information should be collected from the current input and hidden state. The second is the forget gate $\mathbf{f}$, which will learn what information should be discarded from the previous cell state. Finally, the output gate $\mathbf{o}$ regulates the amount of information that goes to the output $\mathbf{h}_t$.

2.8 PyTorch

PyTorch is a deep learning framework for Python that comes with a rich tensor library. It differs from other libraries (TensorFlow, Caffe, Theano and more) in the sense that it builds the computational graph dynamically at runtime. This flexibility allows for controlling the computational flow while training or testing the network.

Appendix A

Appendix

List of Tables

List of Figures

check spelling, capitalization and dates of references

<u>Erklärung</u>

gemäss Art. 28 Abs. 2 RSL 05

Name/Vorname:	....
Matrikelnummer:	....
Studiengang:	…………………………………………………………………………
	Bachelor ☐ Master ☐ Dissertation ☐
Titel der Arbeit:	....
	....
	....
LeiterIn der Arbeit:	....
	....

Ich erkläre hiermit, dass ich diese Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen benutzt habe. Alle Stellen, die wörtlich oder sinngemäss aus Quellen entnommen wurden, habe ich als solche gekennzeichnet. Mir ist bekannt, dass andernfalls der Senat gemäss Artikel 36 Absatz 1 Buchstabe o des Gesetztes vom 5. September 1996 über die Universität zum Entzug des auf Grund dieser Arbeit verliehenen Titels berechtigt ist.

....
Ort/Datum

....
Unterschrift