ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

SCUOLA DI INGEGNERIA E ARCHITETTURA

Dipartimento di Informatica - Scienza e Ingegneria - DISI
Corso di Laurea Magistrale in Ingegneria Informatica

TESI DI LAUREA

in

COMPUTER VISION AND IMAGE PROCESSING

Ambiguity in Recurrent Models: Predicting Multiple Hypotheses with Recurrent Neural Networks

Candidato
Alessandro Berlati

Relatore
Chiar.mo Prof. Luigi Di Stefano

Correlatori
PD Dr. Ing. Federico Tombari[1]
MSc. Oliver Scheel[1,2]

[1]Technische Universität München
[2]BMW Group

Anno Accademico 2017/18
Sessione II

Abstract

Multiple Hypothesis Prediction (MHP) models have been introduced to deal with uncertainty in feedforward neural networks, in particular it has been shown how to easily convert a standard single-prediction neural network into one able to show many feasible outcomes.

Ambiguity, however, is present also in problems where feedback model are needed, such as sequence generation and time series classification. In our work, we propose an extension of MHP to Recurrent Neural Networks (RNNs), especially those consisting of Long Short-Term Memory units. We test the resulting models on both regression and classification problems using public datasets, showing promising results. Our way to build MHP models can be used to retrofit other works, leading the way towards further research.

We can find many possible application scenarios in the autonomous driving environment. For example, trajectory prediction, for humans and cars, or intention classification (e.g. lane change detection) are both tasks where ambiguous situations are frequent.

Prefazione

I modelli a predizione multipla (MHP) sono stati introdotti al fine di gestire l'incertezza nelle reti neurali feedforward, in particolare è stato dimostrato come sia facilmente possibile convertire una rete neurale standard in una in grado di mostrare più risultati plausibili.

L'ambiguità, tuttavia, è presente anche nei problemi in cui sono necessarie reti neurali feedback, come la generazione di sequenze e la classificazione delle serie temporali. Nel nostro lavoro, proponiamo un'estensione di MHP alle reti neurali ricorrenti (RNN), in particolare quelle costituite da celle LSTM (Long Short-Term Memory). Abbiamo testato i modelli risultanti su problemi di regressione e classificazione utilizzando dataset pubblici e mostrando risultati promettenti. Il nostro modo di costruire i modelli MHP può essere utilizzato per riadattare altri lavori, aprendo la strada verso ulteriori sviluppi.

Possiamo trovare molti scenari applicativi nel campo dei veicoli a guida autonoma. Ad esempio, la previsione delle traiettorie di pedoni e auto o la classificazione delle intenzioni (ad esempio il rilevamento del cambio di corsia) sono entrambi compiti in cui situazioni ambigue sono frequenti.

Contents

Chapter 1

Introduction

Time series data are sequences of data points stored in temporal order and represent the evolution over time of a particular system. Time series may represent, for instance, the sequence of logins to a website, the position in the space of an object or scientific measurements.

Time series related problems are a difficult type of task that involve sequence-like inputs. Amongst these problems we find sequence generation, classification, regression and the contexts may be extremely different: from trajectories prediction to financial analysis to weather forecasting.

Time series analysis can be performed with mathematical and statistical instruments, such as linear and non-linear regression or function approximation. However, we are more interested in the context of machine learning and in particular deep learning. In fact, in the last years, Recurrent Neural Networks (RNNs) have been widely used for time series analysis, sequence generation, classification and forecasting with state-of-the-art results in many scientific papers.

In our work we focus on the ambiguity of predictions. Often a neural network becomes overconfident in the outcomes it provides, it does not take into consideration other possibilities that are equally valid. Other times, in uncertain situations, the result may be the average of the possible feasible predictions that is very far from the real one. Imagine we are analysing a video of a person holding a glass, it is possible that this person will start drinking in the immediate future as well as it is possible that he is about to lay down the glass. Both the outcomes are equally valid and building a system able to show the two of them is challenging.

In their work, Rupprecht et al. [48] present a framework that can be used to convert single-prediction systems into multiple-output ones, able to show multiple feasible outputs. We want to extend this framework to RNNs and show that the network is capable of exploring all the feasible possibilities, given a certain input sequence.

In the next chapters we show how it is possible to employ the same formulation as [48], especially the meta-loss function, and build models that can easily retrofit previous work. We focus especially on two problems: trajectory prediction, where we present experimental results on two different models, and sequence classification, where we extend an existing model.

1.1 Thesis Outline

In Chapter 2 we give a brief description of machine learning and neural networks with particular focus on RNNs and their usage for time series prediction and classification.

We then explore other works related to trajectories prediction, time series classification and ambiguity aware models in Chapter 3 and we describe in detail the Multiple Hypothesis Prediction model in Chapter 4, focusing on how a feedforward network can be changed in order to output multiple predictions.

In Chapter 5 we describe the problem and how we want to tackle it, then we present the recurrent multiple hypothesis models that we obtained. The results on real datasets are discussed in Chapter 6.

The thesis in concluded with Chapter 7 where future work is also discussed,

Chapter 2

Background

In this chapter we introduce the fundamental techniques behind the realised work. We begin by describing the foundations of machine learning and deep learning. We then talk about recurrent neural networks and how they are used for time series prediction and classification.

2.1 Machine Learning

Machine learning is the science of getting software to make a determination or prediction generalising from data examples, improving its learning without the need to be explicitly programmed. For instance, imagine we want to handcraft a system that is capable of recognising a dog in an image. Dog breeds are very different from each other. They may differ in size, colour and length of the hair, their muzzle could have different shapes. Also, since we are dealing with images, we need also to consider the different light condition and the pose of the dog. It is basically impossible to implement a system able to consider all these different factors.

In machine learning instead, the program must analyse the training data and find a way to identify dogs. The end goal of machine learning is determining a mapping f between the input X and the output Y, observing a subset $X_{train} \subset X$. The final mapping depends on the metric we use to measure how wrong the prediction is, often called **loss function**, and the techniques employed to minimise this function. We also have different types of problems depending on wether we know the desired output Y or not.

2.1.1 Supervised Learning

When both the input X and the output Y are known we talk about supervised learning. The learning process aims at minimising the error between $y^{(i)} \in Y$ and $\tilde{y}^{(i)} = f(x^{(i)})$ with $x^{(i)} \in X$, the algorithm learns by comparing the output with the target and adjusting the model in order to reduce the error. In this case all the examples must be labeled, in case by hand, therefore it may not be a feasible approach for every problem. Supervised learning is very often used in visual tasks such as image or video classification or object recognition. It is also convenient when historical data are very likely to predict future events. Once the models are trained they can be employed to generate labels for unseen data.
Examples of supervised learning algorithms are: Decision Trees [45], Random Forest [8], linear regression, logistic regression, Support Vector Machines (SVM) [24] and Artificial Neural Networks.

Regression Regression is the supervised learning problem of approximating a function f from input variables X to a continuous real-valued output variable Y. In regression problems a quantity (amount, price, size) is predicted, therefore the model must be evaluated using an error, like the mean squared error. Linear regression is the simplest case of regression, the goal is to find a linear function that maps inputs and outputs. Equation 2.1 is the formalisation of a single-variate case linear regression

$$wx + b = y \tag{2.1}$$

where w and b are the parameter to learn, often called *weight* and *bias*. In Figure 2.1 the dots represent all the training examples while the blue line is the resulting linear model trained on that particular dataset.
Multivariate regression is, however, a more common problem, but the model is straightforwardly derived from the previous one,

$$\sum_{k=0}^{K} w_k x_k + b_k = \mathbf{w}^T \mathbf{x} + b = y \tag{2.2}$$

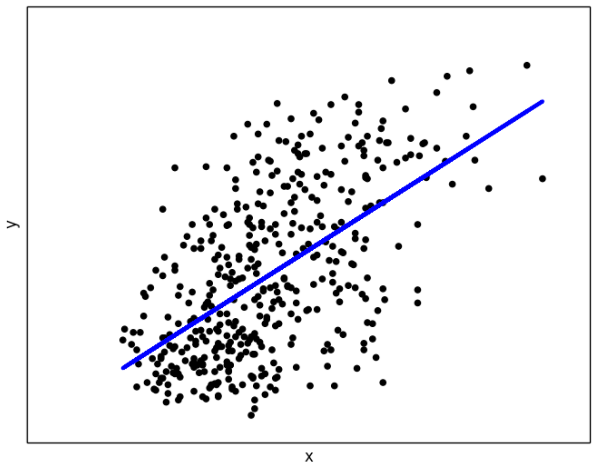


Figure 2.1: Single-variate linear regression

where $\mathbf{w}$ and $\mathbf{y}$ are vectors and b is the sum of al the biases. From Equation 2.3 we can also model a multi-variate output regression by stacking a single output formulation for every output value, in matrix form:

$$\mathbf{W}^T\mathbf{x} + \mathbf{b} = \mathbf{y} \tag{2.3}$$

Classification A classification problem aims at approximating a function f mapping input variables X to discrete output variables Y that are called classes, labels or categories. Classification models can predict the class directly (e.g. random forests) or continuous values that are then converted into probabilities. In the last case, the class with the highest probability is the predicted label. Since the output is discrete, the classification **accuracy** can be computed and used as evaluation measure for the model. There are also other indicators such as precision and recall (explained later). The labels are commonly represented by one-hot encoding, a sparse binary vector with a 1 in the i-th position to indicate the actual class; an example has usually only one class.

Logistic Regression can be used to tackle problems with only two classes, also called binary classification problems, such problems include spam recognition, disease detection or quality control (pass/fail) test. The resulting class must be in the $[0, 1]$ range, where a higher value represents a higher probability of the input belonging to the class. The linear model (Equation 2.1) result, that we will call z, must then be bounded in the probability range; to do so we can use a sigmoid function:

$$\sigma(z) = \frac{e^z}{1 + e^z} \tag{2.4}$$

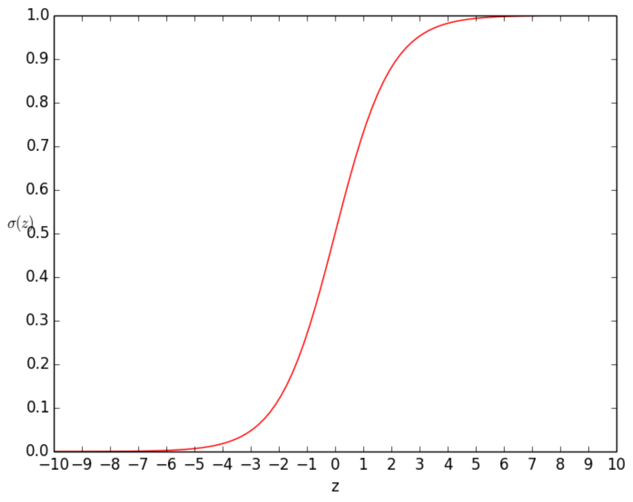


Figure 2.2: Sigmoid function

As we can see in Figure 2.2 if z is high the probability tends towards 1 while when z is low towards 0.

In case we need to predict multiple classes, we could train one different classifier for every class with logistic regression and then choose the one with the highest probability; this strategy is called One-versus-All (OvA). An easiest approach consists in using the **softmax** function that transforms a K-dimensional labels vector $\mathbf{z}$ of arbitrary real values into a K-dimensional vector with each value bounded between 0 and 1, additionally the sum of all the entries will be exactly one:

$$softmax(z)_k = \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}}, for \;\; k = 1, ..., K \tag{2.5}$$

2.1.2 Unsupervised Learning

Unsupervised learning is used to infer a function that describes the structure of unlabelled data. The aim of unsupervised learning is obtaining more information on the input, learn more about it, for this reason it is difficult to define a correct answer. Problems like these include:

- **Cluster Analysis** which is used to explore the examples in order to find hidden patterns or grouping in data, for instance market researchers may use clustering to group customers into different market segments.
- **Association Rule Learning** used to discovers rules that describe large portions of your data, this could be particularly useful for decision about marketing activities such as pricing and product placement.

2.2 Neural Networks

Artificial Neural Networks (ANN) are computing systems composed of simple, connected components that are inspired by the way *biological* neural networks work. For this reason the nodes that make up a NN are called **artificial neurones**. The simplest form of NN is the Multilayer Perceptron (MLP) and it is formed by an input layer and at least two layers of neurones (also called perceptrons). MLP belongs to the so called *feed-forward* networks, where the information flows in only one direction and they are therefore characterised by the absence of loops. Initially NNs had only one hidden layer, in fact the **universal approximation theorem** [26] states that three-layer networks with a finite number of neurones could approximate every continuous function.

However, to do so, a prohibitive number of neurones is required, making the training impossible. The *deep learning* paradigm has been introduced to solve the *shallow networks* problem, Deep Networks have a high number of layers with less neurones, therefore they can approximate complex functions with a smaller number of learnable parameters. In deep networks every layer learns a different function, deep layers abstract high level functions that combined together can produce very complex systems.

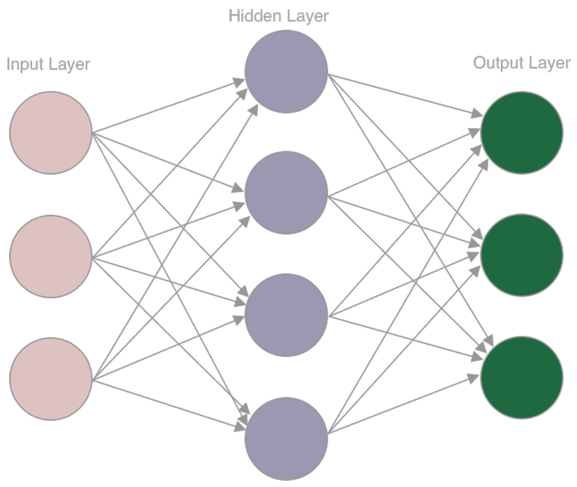


Figure 2.3: A three layers MLP

A perceptron is defined as follows:

$$y = \varphi\left(\sum_{i=1}^{n} w_i x_i\right) \tag{2.6}$$

where x_i is the *i-th* input, w_i is the associated weight, y is the neurone output value while φ is the activation function.

Activation functions can be different amongst layers, we can therefore see deep networks as compositions of functions that can be linear or non-linear. However the composition of linear functions is a linear function as well and cannot model non-linear behaviours, which is what hidden layers are used for. For this reason the most used activation functions are non-linear, such as: Sigmoid, ReLU, Leaky ReLU, ELU.

The basic layer in deep networks is the **fully connected** layer. In this layer, the output of every neurone depends on all the values from the previous layer that are usually summed and passed through a non-linear activation function. Fully connected layer however have shortcomings:

- Highly dimensional inputs make the number of learnable parameters explode
- Spatial information is not taken into consideration

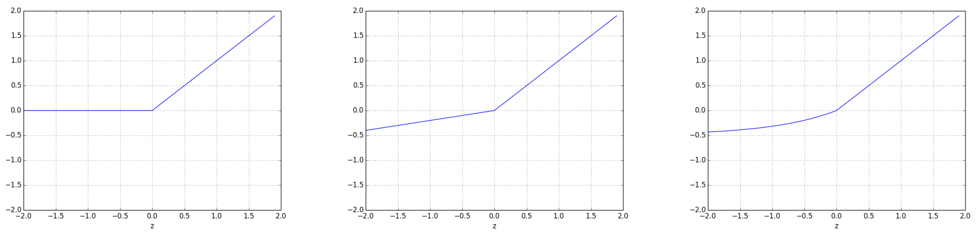

(a) ReLU (b) Leaky ReLU (c) ELU

Figure 2.4: Activation Functions

Both these problem are present in case images are used as input. **Convolutional** layers are used to overcome the limitations of fully connected layers. Let us analyse a two dimensional convolution. Let the input image x have size WxH and a squared convolutional filter have size NxM. The element $y_{i,j}$ of the output y, considering $N = M = 3$, is defined as follows:

$$\begin{aligned} y_{i,j} = {} & w_{0,0}x_{i-1,j-1} && +w_{1,0}x_{i-1,j} && +w_{2,0}x_{i-1,j+1} && + \\ & w_{0,1}x_{i,j-1} && +w_{1,1}x_{i,j} && +w_{2,1}x_{i,j+1} && + \\ & w_{0,2}x_{i+1,j-1} && +w_{1,2}x_{i+1,j} && +w_{2,2}x_{i+1,j+1} && +b \end{aligned} \tag{2.7}$$

The same filter is used to produce the whole output y so a fixed number $(N * M)$ of learnable parameters is required. Also spatial information is considered since close inputs are processed together. Convolutions can also be one dimensional, with N learnable parameters, and two dimensional with channels, with $N * M * C$ parameters, to process multichannel inputs (i.e. RGB images).

A convolutional layer is composed of one or more multichannel convolutional filters, resulting in $N*M*C*D$ learnable parameters and a multichannel two dimensional output. To attain non-linearity every output element $y_{i,j}$ is activated with a non-linear activation function.

2.2.1 Parameter Optimisation

The training of a model aims at modifying the parameters θ in order to minimise the error between the predictions $\tilde{y}$ and the ground truth y. The error is usually defined using a loss function $\mathcal{L}(\tilde{y}, y)$ that can be different depending on the task. Largely used loss functions are:

- **Mean Absolute Error** $\mathcal{L}_{MAE}(\tilde{y}, y) = \frac{1}{n}\sum_{k=0}^{K} |y_k - \tilde{y}_k|$
- **Mean Squared Error** $\mathcal{L}_{MSE}(\tilde{y}, y) = \frac{1}{n}\sum_{k=0}^{K} (y_k - \tilde{y}_k)^2$
- **Categorical Cross Entropy** $\mathcal{L}_{cross} = -\sum_{k=0}^{K} y_k log(\tilde{y}_i)$

The parameters θ of a model f are considered optimal when the loss over all the N examples is minimal:

$$\theta_{opt} = arg\min_{\theta} \sum_{i=0}^{N} \mathcal{L}(f(x^{(i)}; \theta),\, y^{(i)}) \tag{2.8}$$

Gradient descent is an optimisation algorithm used to find the local minimum of a function. It is based on the fact that a function $F(x)$, defined and differentiable in a neighbourhood of x_0, decreases fastest in the direction of the negative gradient $-\nabla F(x_0)$. This could be applied to the neural network in order to update the parameters θ one step at a time towards the optimum. The neural network update step can be formalised as:

$$\theta_{t+1} = \theta_t - \gamma \nabla_{\theta_t} \left(\frac{1}{N} \sum_{i=0}^{N} \mathcal{L}(f(x^{(i)}; \theta),\, y^{(i)}) \right) \tag{2.9}$$

where γ is the *learning rate*, a factor that influence the training speed and accuracy. Larger learning rates yield bigger update steps decreasing the time needed to reach convergence at the expense of the precision. Often learning rate is dynamically decreased over time.

Computing the gradient of every weight in a neural network is however not trivial, for this **Backpropagation** is used. Backpropagation gives a fast way to compute all the partial derivatives $\partial f / w_i$, starting from the gradient of the loss, using the chain rule of differentiation: $\frac{\partial y}{\partial x} = \frac{\partial y}{\partial u}\frac{\partial u}{\partial x}$ with $y = f(g(x))$, $u = g(x)$, u differentiable in x and y differentiable in $g(x)$. The set of partial derivatives can be then used to perform a gradient descent update step.

Gradient Descend has one big limitation, it must see the whole dataset before making one update step, it is consequently unfeasible to use with big datasets. Stochastic Gradient Descent (SGD) has been introduced to overcome this problem, it is capable of updating the learning parameters just analysing a batch of the training data. After SGD many improvements has been made, such as Momentum, and other algorithm, based on gradient descent, has been introduced such as AdaGrad [16], RMSProp and ADAM [31].

2.3 Recurrent Neural Networks

Recurrent neural networks (RNNs) are a family of neural networks for processing sequential data. Convolutional neural networks are specialised in processing grid-like values, such as images, where the spatial information is available and useful. RNNs are instead used to process sequences of values. Like CNNs share the same parameters across all the values, RNNs share the same weights across several time steps. Imagine you want to generate some text, it is important to consider the words that come before, the context of the sentence. If you are predicting one character at a time you must take into consideration the possible combination of letters and the punctuation. An RNN is able to remember what it has already processed, the output is therefore derived from what happened before. This is how a RNN typically looks like:

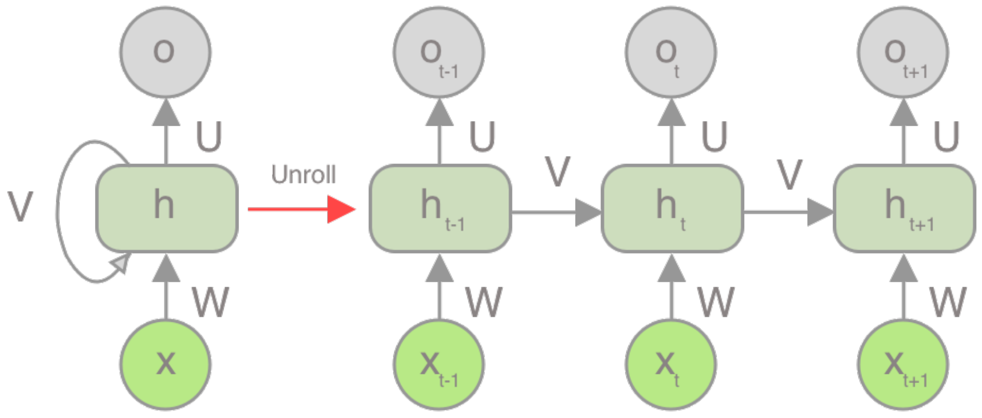


Figure 2.5: Recurrent Neural Network

Since RNNs have loops they are classified as *feedback* networks, this allows

them to remember the inputs they already processed. They store information in the hidden state which is shared amongst all the time-steps. These loops may seem confusing at first, but if the network is unrolled (wrote out for the complete sequence) it looks very similar to a standard neural network in which every layer takes two inputs, one from the previous layer (hidden state) and the traditional input .
The hidden state is the memory of the network and it is usually computed by means of a non-linear function, such as an hyperbolic tangent (*tanh*), and it is calculated based on the current input and the previous state; the output is instead computed just based on the current state.
RNNs can be trained with the backpropagation algorithm, similarly to traditional Neural Networks, but to do so it is necessary to consider the temporal nature of the problem. Since the parameters are shared across all time-steps in the network, the gradient of the error must be computed not only on the current result but also on all the previous time-steps. For example, in order to calculate the gradient at t=10 we would need to backpropagate ten steps and sum up the gradients.

2.3.1 Vanilla RNN

In a Vanilla Recurrent Network the repeating module has a very simple structure, such as a single *tanh* layer. The current state is then computed as a function of the previous state h_{t-1} and the current input x_t.

$$h_t = tanh(W_t * h_{t-1} + W_x * x_t) \tag{2.10}$$

Once obtained the current state we can compute the output as:

$$y_t = W_y * h_t \tag{2.11}$$

Gradient Problem Simple (Vanilla) RNNs have difficulties learning long-term dependencies due to what is called the vanishing/exploding gradient problem. We consider L as our loss (error) function, for example cross-entropy in a classification problem, it can be computed based on targets and prediction. The gradients of the error must then be computed to perform the weight update. Following the Backpropagation algorithm, we can notice

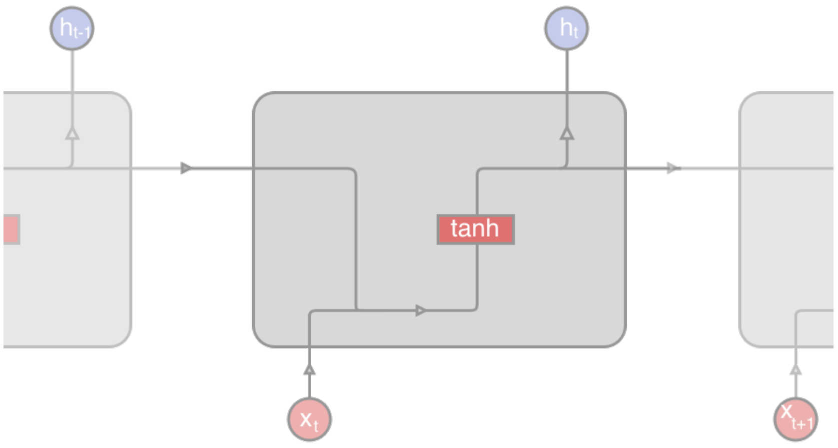
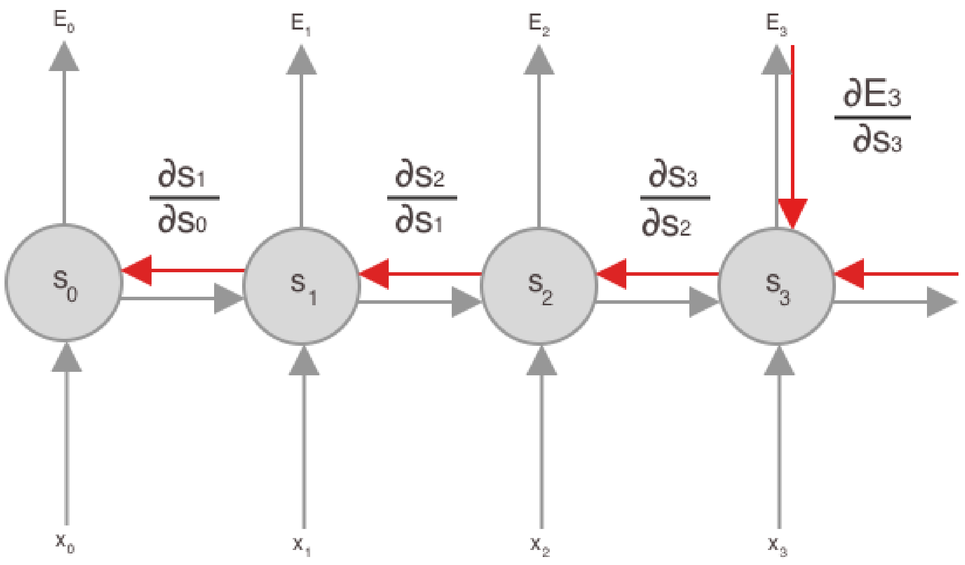


Figure 2.6: Vanilla RNN

that to reach the end of the graph, that in this case is the unrolled RNN, we must consider all the previous time-steps (as showed in Figure 2.7) applying the chain rule of differentiation. This process is very similar to standard BP but it is called Backpropagation Through Time (BPTT) to stress how the gradients are actually computed.

1. Exploding Gradient: In RNNs (or deep networks in general), error gradients can accumulate and become very large numbers resulting in big updates to the network weights. This may eventually bring instability to the network. At an extreme, the weights values can become so large as to overflow and result in NaN values. The explosion occurs by repeatedly multiplying gradients through the network layers that have values larger than 1.0. In recurrent neural networks, exploding gradients can result in a network that is completely unable to learn from training data and at best a network that cannot learn over long input sequences of data. This problem is usually solved using gradient clipping, limiting the norm of the gradients to a certain value.
2. Vanishing Gradient: It is the opposite problem but it follows the same mechanism. If the gradient values are small the final gradient will shrink exponentially fast, eventually vanishing completely after a few



E_0
E_1
E_2
E_3
$\frac{\partial E_3}{\partial s_3}$
$\frac{\partial s_1}{\partial s_0}$
$\frac{\partial s_2}{\partial s_1}$
$\frac{\partial s_3}{\partial s_2}$
s_0
s_1
s_2
s_3
x_0
x_1
x_2
x_3

Figure 2.7: Backpropagation Through Time

time steps. Gradient contributions from farther steps become zero, basically the state at those moments does not contribute to what the network is learning, long-term dependencies are not learned. Vanishing gradients also happen in deep feedforward Neural Networks but in RNNs the problem is more common since they tend to be deeper.

These problems, that are deeply discussed [40], are not only present in feedback networks but they belong to deep networks in general. However the RNNs depth depends on the time-steps already processed and therefore gradient problems are easier to encounter.

2.3.2 Long Short-Term Memory Networks

Long Short-Term Memory networks (LSTM) are a particular type of RNN, capable of learning long-term dependencies. They were designed by Hochreiter and Schmidhuber [25] to avoid the long-term dependency problem. Consequently LSTM are now extensively used on a large assortment of problems. LSTMs have the same chain structure as vanilla RNNs but the repeating module has a more complex structure, four functions interacting instead of just one.

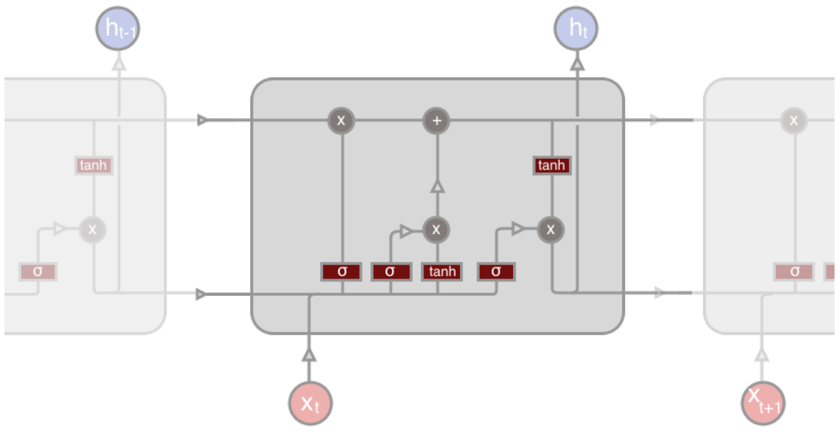


Figure 2.8: Long Short-Term Memory Cell

The key to LSTMs is the cell state, computed following the horizontal line running through the top of the diagram. The cell state is one of the two hidden states that are passed through time. Using particular structures, called gates, the LSTM has the possibility to remove or add information to the cell state. Gates are usually composed of a sigmoid function and a point-wise multiplication. There are three of these gates in a standard LSTM cell:

- Forget Gate: takes h_{t-1} and x_t as input and outputs a number between 0 and 1 for each component of the cell state C_{t-1}. A 1 keeps the value while a 0 forgets it.

$$f_t = \sigma(W_f * [h_{t-1}, x_t] + b_f) \tag{2.12}$$

- Input Gate: used to update the values of the cell state, combined to a vector of new candidate values $\tilde{C}_t$.

$$i_t = \sigma(W_i * [h_{t-1}, x_t] + b_i) \tag{2.13}$$

$$\tilde{C}_t = tanh(W_C * [h_{t-1}, x_t] + b_C) \tag{2.14}$$

- Output Gate: in the same way as the other gates decides what the network is going to output. It will be then combined with the current state.
$$o_t = \sigma(W_o * [h_{t-1}, x_t] + b_o) \tag{2.15}$$

The new Cell State C_t and the new hidden state h_y are computed as follows
$$C_t = f_t * C_{t-1} + i_t * \tilde{C_t} \tag{2.16}$$
$$h_t = o_t * tanh(C_t) \tag{2.17}$$

LSTMs and variation of this concept solve the vanishing gradient problem by eliminating the activation function for the cell state. The activation function is the identity which leads to a gradient of 1, removing all the possible problems.

Chapter 3

Related Work

Our work is mainly focused on sequence prediction and classification. We are particularly interested in predicting trajectories of moving objects. Standard mathematical methods have been used for path prediction, amongst these we find linear regression [37] , autoregressive models [3], Kalman filters [6], time-series analysis [43] and non-linear regressions [52, 51, 46, 44].

Other papers use Recurrent Neural Networks for enhancing trajectory prediction. RNNs, in fact, have been already successfully employed for sequence-related task such as speech recognition [20, 10, 12, 21], image captioning [29, 53, 14, 30], video classification [38], human motion estimation [41, 17] and machine translation [5, 9] .

In their work, Graves et al. [19] used an LSTM network for sequence generation. In their first experiment they train the network to predict text one character at a time. They test the network on two datasets, Penntree Bank [36] and the Hutter Prize portion of English Wikipedia [27], demonstrating that LSTMs are capable of learning a large vocabulary of words as well as inventing english-looking words and names. Also, they showed the network's capability of using punctuation, for example opening and closing parenthesis, and formatting marks, such as XML tags. In the second experiment they test the network's ability to predict real-valued sequences, in particular online handwriting. The input data is a sequence of (x, y) pen coordinates but the network is trained to predict the parameters of a mixture of bivariate Gaussians (mean, standard deviation and correlation for both coordinates).

The results show generated letters, strokes and short words and the ability to generate long coherent sequences.

Alahi et al. [4] propose an Encoder-Decoder LSTM model for human trajectory prediction in crowded spaces. Since humans adapt their movements also based on the people they have around, they introduce a model that take into consideration the neighbouring agents. The network consists in a LSTM cell for each person in the scene and a pooling layer used to condense and combine the hidden states of the people that are close together. The hidden states are thought to contain information about the motion properties of an agent with respect to its surroundings. Then, at every time-step, the LSTM cell is fed with the pooled information to predict a new position. Lee et al. [34] propose an Encoder-Decoder architecture, using Gated Recurrent Units (GRU [11]), to predict socially acceptable trajectories that take into consideration the path intrinsic multi-modality, using a conditional variational auto-encoder (CVAE [9]), and the context, by means of a CNN. Gupta et al. [22] introduce a combination of LSTM Encoder-Decoder and Generative Adversarial Networks (GANs), a variety loss function allows them to explore more than one hypothesis while the GAN will choose the best amongst them. This last paper also improved the pooling layer from [4] in terms of speed and simplicity, they use a Multi Layer Perceptron (MLP) followed by max pooling instead of computing an occupation map for every agent and then concatenate the hidden states.

Another interesting problem for sequence classification is intention prediction for Advanced Driver Assistance Systems (ADAS). These systems alert the driver in case of dangerous manoeuvres, therefore anticipating the driver intentions could prevent accidents. This problem has been treated by Tran et al. [50] using Hidden Markov Models but also using RNNs as in the case of Jain et al. [28] and Scheel et al. [49], both these last two papers treated the lane change problem which we will consider as well.

There are many works that address the uncertainty in neural networks and the multiple prediction problem. Amongst the most famous models we can find Mixture Density Networks [7] (MDNs) that are a particular type of neu-

ral networks that predict the parameters of a Gaussian Mixture model. If we sample multiple times using the same parameters we can generate different hypothesis. This model however works only in regression problems and it lacks precision when high dimensional output are needed. Geng et al. [18] deal with the ambiguity of age estimation introducing a label distribution for every image, in this way a single label can influence also the learning of other adjacent labels. The same problem, along with image classification, is treated by Gao et al [18].
Multiple Choice Learning [23] (MCL) paradigm aims at minimising a loss over the outputs of a set of predictors. MCL has been exploited in [13, 33] and by Lee et al. [35] where they train a set of deep networks with a architecture and loss agnostic method. Rupprecht et al. [48] introduced a new way to deal with the uncertainty of Neural Networks predictions, their framework allows to easily convert a feedforward neural network into a multiple prediction version (see Chapter 4) that is capable of exploring more than one feasible outcomes in an ambiguous context. Their method can be applied on any feedforward architecture and there is not the necessity to train multiple networks, reducing the number of parameters with respect to MCL. Moreover they present a general meta-loss with which training can be performed using standard backpropagation and they also provide a mathematical explanation of the benefits using this approach. In our work we want to extend this last approach to RNNs. Exploiting their method we try to generate and classify sequences in ambiguous situations, studying if feedback networks are able to explore multiple feasible results.

Chapter 4

Multiple Hypothesis Prediction

Ambiguity is present in every kind of task. We may encounter uncertainty in object classification problems, where salient objects are not labeled, or in pose estimation, where ambiguous values are often predicted for occluded elements. Future prediction is especially difficult since the task itself is uncertain and many outcomes can be equally right. In their work, Rupprecht et al. [48] propose a general framework for Multiple Hypothesis Prediction (MHP) that allows to convert any single-prediction CNN architecture into a multiple-output one. The original loss is wrapped into a *novel* meta-loss so that training can be achieved by gradient descend and backpropagation. It is shown that minimising the new problem generates a Voronoi tessellation of the output space based on the problem loss.

Definition 1. ***Voronoi Tessellation*** *Let X be a metric space with a distance function d and let $P = p_1, ..., p_k$ be a set of sites in X. A Voronoi region, or Voronoi cell, C_k associated with the site P_k is defined as follows:*

$$C_k = \{x \in X \mid d(x, P_k) \leq d(x, P_j) \; \forall j \neq k\}$$

In other words a Voronoi cell is the set of all points whose distance from the associated site is less than the distance to any other site. The Voronoi Tessellation of X is therefore the collection of the Voronoi cells associated to a given set of sites.

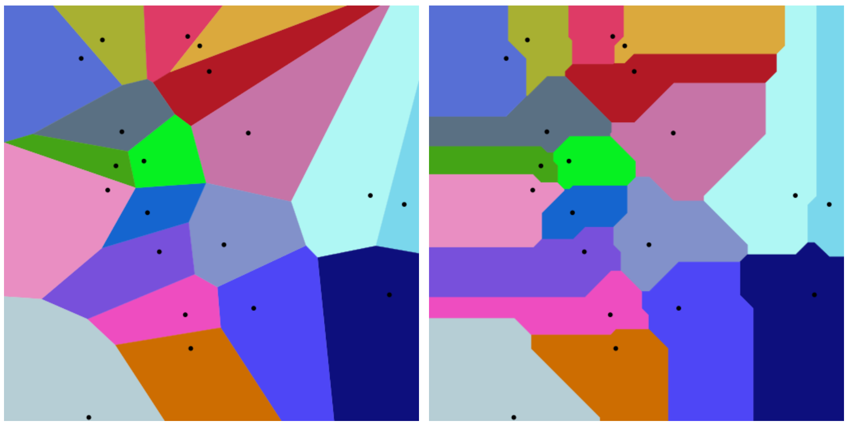

(a) Euclidean distance [2] (b) Manhattan distance [1]

Figure 4.1: Voronoi tessellation using two different distances

4.1 Derivation of MHP Model

In this section we will explain the derivation of the ambiguity-aware model from the traditional single hypothesis model. The vector space of input variables will be indicated as $\mathcal{X}$ and the vector space of output variable as $\mathcal{Y}$, the joint probability density over input variables and output will be represented by $p(x, y)$. We also assume to have a training set made of N tuples (x_i, y_i), where $i = 1, ..., N$

4.1.1 Single Hypothesis Model

Training a predictor $f_\theta : \mathcal{X} \rightarrow \mathcal{Y}$, parametrised by $\theta \in \mathbb{R}^n$, in a supervised scenario we aim at minimising the expected error:

$$\frac{1}{N}\sum_{i=1}^{N} \mathcal{L}(f_\theta(x_i),\, y_i) \tag{4.1}$$

In case N is large enough, Equation 4.1 provides a good approximation of the continuous error

$$\int_{\mathcal{X}}\int_{\mathcal{Y}} \mathcal{L}(f_\theta(x), y)p(x, y)\; dy\; dx \tag{4.2}$$

that is minimised by the conditional average (see [32])

$$f_\theta(x) = \int_{\mathcal{Y}} y \cdot p(y|x)\, dy \tag{4.3}$$

The conditional average is not always a good representation, in fact the expected value may fall where the conditional probability density $p(y|x)$ is low. Imagine a system that wants to predict a value that is randomly 0 or 256, the average output will be 128. In these cases, using a model capable of predicting multiple feasible outcomes may be a better approach.

4.1.2 Multiple Hypothesis Model

Let us consider the Voronoi tessellation of the output space $\mathcal{Y} = \cup_{i=1}^{M} \overline{\mathcal{Y}_i}$ induced by M generators (sites) $g^i(x)$ and the loss $\mathcal{L}$ associated with the problem. Every cell $\overline{\mathcal{Y}_i}$ contains all the points that are closest to the site $g^i(x)$. In our case the vicinity depends on the loss function $\mathcal{L}$.

$$\mathcal{Y}_j(x) = \{y \in \mathcal{Y} : \mathcal{L}(g^j(x), y) < \mathcal{L}(g^k(x), y)\ \forall k \neq j\} \tag{4.4}$$

The ambiguity aware model, proposed in the paper, is composed by a single predictor, instead of multiple ones, able to output M values,

$$f_\theta(x) = (f_\theta^1(x), ..., f_\theta^M(x)) \tag{4.5}$$

and the loss $\mathcal{L}$ is always computed for the closest of the M predictions.

$$\int_{\mathcal{X}} \sum_{j=1}^{M} \int_{\mathcal{Y}_j(x)} \mathcal{L}(f_\theta^j(x), y) p(x, y)\, dy\, dx \tag{4.6}$$

Therefore, training such model in order to minimise Equation 4.6 will induce a Voronoi tessellation of $\mathcal{Y}$ in M cells, generated by the predicted hypotheses f_θ^i. In a regression problem $\mathcal{L}$ is typically the l_2-loss. In case the density $p(x, y)$ satisfies mild regularity condition, it is equal to zero only for a null subset, then the following theorem is valid.

Theorem 1. ***(Minimiser of 4.6)*** *In order for Equation 4.6 to be minimal it is necessary that the sites $g^i(x)$ are identical to the predictors f_θ^j, an both* *correspond to a centroidal Voronoi tesselation:*

$$g^j(x) = f^j_\theta(x) = \frac{\int_{\mathcal{Y}_j} \mathcal{L}(f^j_\theta(x), y) p(y|x)\, dy}{\int_{\mathcal{Y}_j} p(y|x)\, dy} \tag{4.7}$$

i.e f^j_θ predicts the conditional average of the Voronoi cell it defines.

Proof can be found in [48] and [15]

4.2 Minimisation Scheme

In this section we explain how Equation 4.6 can be minimised, using a neural network and a generic dataset composed of N examples. The algorithm can be explained in a few steps but we must take into consideration that we are training the network with back-propagation:

1. As explained in Theorem 1, it is necessary to create M sites that correspond to the predictions $f^j_\theta(x_i), j \in \{1, ..., M\}$. This is done for each training sample (x_i, y_i) by a forward pass through the neural network.

2. Using the chosen loss function $\mathcal{L}$, the examples (x_i, y_i) and the sites/predictions $f^j_\theta(x_i)$, build the Voronoi tessellation of $\mathcal{Y}$.

3. Each Voronoi cell must be used to compute the back-propagation gradients of the error $\frac{\partial}{\partial \theta} \frac{1}{|\mathcal{Y}_i|} \sum_{y_i \in \mathcal{Y}_i} \mathcal{L}(f^j_\theta(x_i), y_i)$, with $|\mathcal{Y}_i|$ cardinality of $\mathcal{Y}_i$. If the example belongs to the cell it contributes to the error.

4. An update step is now performed taking into consideration the gradient per hypothesis computed in the previous step. At this point, if a convergence criterion is not met we continue with Step 1.

If we consider the meta-loss in Equation 4.8, we can implement the algorithm directly and without particular difficulties.

$$\mathcal{M}(f_\theta(x_i), y_i) = \sum_{j=1}^{M} \delta(y_i \in \mathcal{Y}_j(x_i)) \mathcal{L}(f^j_\theta(x_i), y_i) \tag{4.8}$$

The easiest way to convert an existing network into a MHP one is to copy the output layer multiple times (before the initialisation). In this way it is possible to generate multiple answers. At training time, for each example we compute the loss on all the hypotheses. The one providing the lowest error is used in the meta-loss. Since the predictors may be initialised very far from the labels y, it is possible that all these lie in a single Voronoi cell and therefore only one predictor gets the updates. To avoid this problem it is necessary to relax the Kronecker delta δ: from a hard 0-1 assignment to

$$\hat{\delta}(c) = \begin{cases} 1 - \epsilon & \text{if } c \text{ is true} \\ \frac{\epsilon}{M-1} & \text{else} \end{cases} \tag{4.9}$$

In this way the closest hypothesis gets a label y and contributes to the loss with a high weight and all the others with a small one. This framework allows us to take an existing feedforward network (e.g CNN) with any loss, embed the loss in Equation 4.8, replicate the output layer and train the network to output multiple predictions.

The experiments show that MHP models outperform the associated SHP ones. MHP models can also be used to provide the variance over the hypotheses that is an additional information to assess if a situation is ambiguous or not.

Chapter 5

Problem, Methodology, Rnn extension and model(s)

In this chapter we first describe the ambiguity problem in trajectory prediction. In Section 5.1 we provide some examples with standard neural networks to present the issue and how predicting multiple hypotheses can help tackle it.
In Section 5.2 we talk about extending MHP to RNN models in order to predict multiple trajectories, while in Section 5.3 we display the multiple classification problem and the relative MHP model.

5.1 Ambiguity in Trajectory Prediction

To explain the problem we consider a situation in which a car is approaching an intersection (or equivalently a roundabout), there are three possible outcomes: the car could proceed straight, it could turn right or it could turn left. A single hypothesis neural network predicts always the most probable outcome, but in case the car didn't give any hint on the manoeuvre the result may have about the same probability as the other possibilities, in case of classification, or might be an average trajectory in case of prediction. In both cases the answer that the network returns in an ambiguous situation is not satisfactory.

We try to reproduce this problem with a toy example. To do so we use a

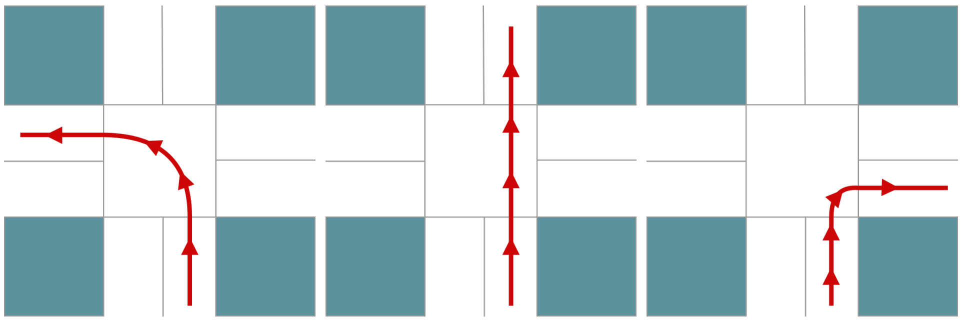

(a) Left (b) Straight (c) Right

Figure 5.1: Intersection example trajectories

handmade dataset composed of trajectories, sequences of x and y real-valued tuples, resembling a car approaching an intersection, like the ones we see in Figure 5.1. We then model two problems:

- **Classification**: given a point the network should predict the end direction of the object, the three possible labels are Right, Left and Straight.
- **Regression**: given a point the network should predict the endpoint, x and y coordinates, of the trajectory.

We associate a label to every point, points belonging to the same trajectory share the same label, and train a simple neural network with one hidden layer. The obtained results are consistent with our assumptions, when the input represents an ambiguous situation the neural network does not give a reliable answer. In the classification case, as shown in Table 5.1, the model gives a safe answer for the first two cases but in the third one the answer is not reliable, as well as wrong, as the probability is just slightly higher than the other two possibilities.

Similarly, in the regression case (Figure 5.2) the model outputs a very accurate endpoint for unambiguous inputs, while it returns a point in the middle of the intersection in the last case. When the input may produce different feasible outcomes the model tends to return the average between them, resulting in a point very distant from ground truth.
It is clear that a model able to provide multiple answers may prove very useful in exploring different feasible results and therefore be more accurate.

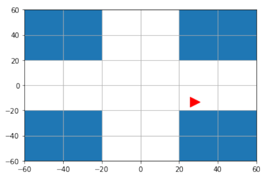
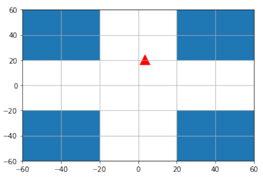
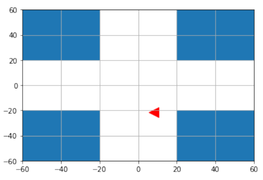
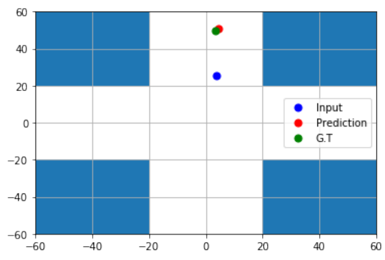
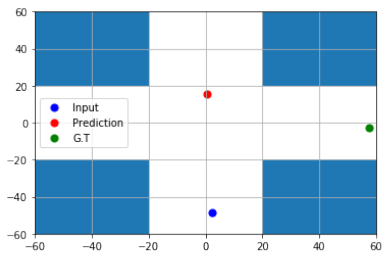
	Right	Straight	Left
			
Left	0%	0%	33.36%
Straight	0%	**100%**	31.69%
Right	**100%**	0%	**34.95%**

Table 5.1: SHP Model Output Class Probabilities

We already know from Chapter 4 that it is sufficient to replicate the output layer as many times as the number of hypotheses we want, with random initialisation, to obtain an MHP version of the network. This can be further trained embedding the previous loss in the MHP meta-loss (Equation REF).



(a) Non ambiguous

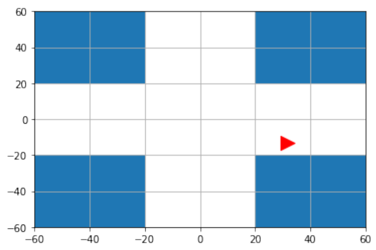


(b) Ambiguous

Figure 5.2: SHP Model Regression Results - TODO

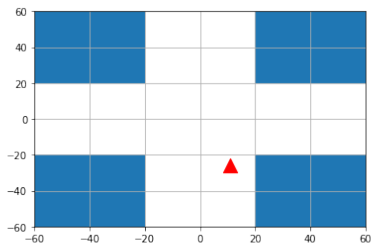
The MHP classification results show that all the hypotheses return the same class in the non ambiguous situation (Table 5.2a) while in the other case (Table 5.2b) every prediction is different. The difference between SHP and MHP here is clear: the SHP model choses the highest class amongst very similar probabilities, the MHP model understands the uncertainty and every hypothesis gives a feasible answer with 100% probability.

The contrast between MHP and SHP in regression (Figure 5.3b) is even



(a) Ground Truth: Left

	Left	Straight	Right
#1	**100%**	0%	0%
#2	**100%**	0%	0%
#3	**100%**	0%	0%

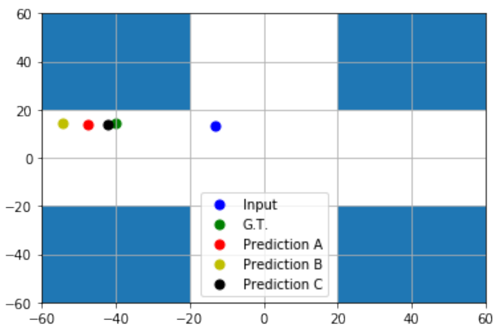


(b) Ground Truth: Straight

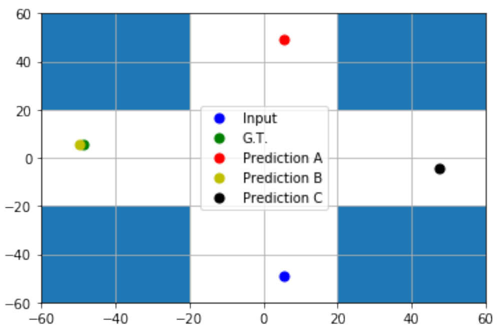
	Left	Straight	Right
#1	2%	2%	**95%**
#2	**94%**	3%	3%
#3	3%	**95%**	2%

Table 5.2: Classification per Hypothesis

more evident. In the ambiguous case, instead of predicting the mean point, every hypothesis return a feasible endpoint. We may not know which one is the more plausible but we know that one of these is very close to the ground truth. As in classification, in an unambiguous situation all the hypothesis are similar, showing no uncertainty.



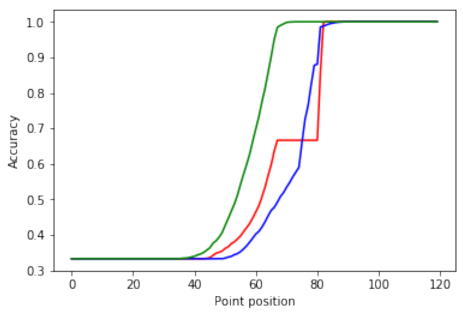
(a) Non Ambiguous



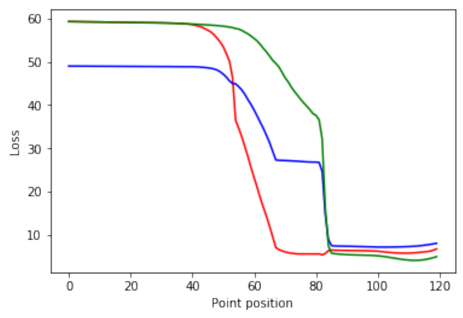
(b) Ambiguous

Figure 5.3: MHP Regression - Blue input, Green Ground Truth, Other Predictions

The model does not take into consideration the temporal position of the input point in the whole trajectory, but we can still show how the accuracy/loss vary with respect to that. In Figure 5.4 we can notice how the result improve if the processed point is closer to the end of the sequence, the model is more accurate when the input is non ambiguous.



(a) Classification Accuracy



(b) Regression Loss

Figure 5.4: Accuracy and Loss related to the Point position

These simple experiments prove that MHP could be successively used in this context, leading to more accurate predictions in ambiguous cases.

5.2 Multiple Trajectory Prediction: MHP Extension to RNNs

In this section we focus on a particular type of time series: trajectories of objects. We want to create a recurrent architecture capable of predicting multiple feasible future trajectories, given the sequence of previous positions as input. To do so we want to apply the MHP method. Our goal is to prove that MHP could be applied to recurrent models, therefore we omit everything related to the spatial context and the social acceptability of the predictions in order to simplify as much as possible the models. Since state of the art papers [4, 22] take into consideration these information thanks to an encoder-decoder architecture we decided to use that as well, in this way it should be easier to retrofit previous single-prediction models.

Problem formulation We assume to have N trajectories composed of T ij-coordinates (i_t^n, j_t^n), representing an object position in the space, for T subsequent time-steps. We receive as input the position of an object from time 0 to T_{in} and we try to predict multiple sequences of positions from time T_{in+1} to T. The goal is to obtain multiple feasible trajectories. In case the input defines an ambiguous situation the predictions should show the different possible outcomes; they should instead be similar in the opposite case. We define the n-th input trajectory as $X^n = (x_t^n) = (i_t^n, j_t^n)$ with $t = 0..T_{in}$ and the correspondent output trajectory as $Y^n = (y_t^n) = (i_t^n, j_t^n)$ with $t = 0...T_{pred}$. We represent the K predictions as $\tilde{\mathbf{Y}}^n$ and the *k-th* hypothesis as $\tilde{Y}_k^n$.

Model 1 As already stated in Chapter 4, in order to transform a feedforward single-prediction model into an MHP model it is sufficient to replicate the output layer. We want to apply this definition also to LSTMs, also maintaining an encoder-decoder architecture.
The first model is composed of three parts. The first part is an LSTM encoder that process the input trajectory X^n and produces an hidden state $S^n_{T_{in}}$. The internal state is then used as *initial state* for the LSTM decoder to generate a sequence of high dimensional points Z^n where the number of dimensions depends on the size of the LSTM. Every element of Z^n is then transformed, through a fully connected layer, into K different (i, j) tuples. Together, the sequences of tuples together form the different output trajectories.

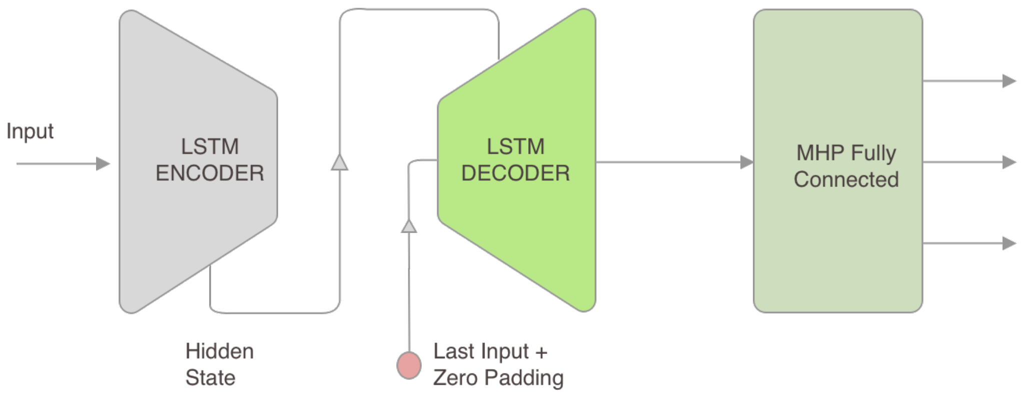


Figure 5.5: Model 1

The key idea behind this model is to produce high dimensional representation of the output space that can be converted into different tuples of coordinates. The network must learn how to represent ambiguous situations through the hidden state and also how to extract the points from the decoder output representation.

Model 2 In the second model we want to generate directly the different trajectories. To do so it is necessary to run the decoder multiple times with different initialisation. We maintain the LSTM encoder to process the input sequence and get the resulting hidden state $S^n_{T_{in}}$. Then, we employ a fully connected layer to generate K different states $S^n_{T_{in},h}$. At this point we can run the decoder multiple times with distinct initial states generating the output sequences. Finally, we exploit a linear transformation to convert the LSTM output points into coordinates, thus obtaining the trajectories.

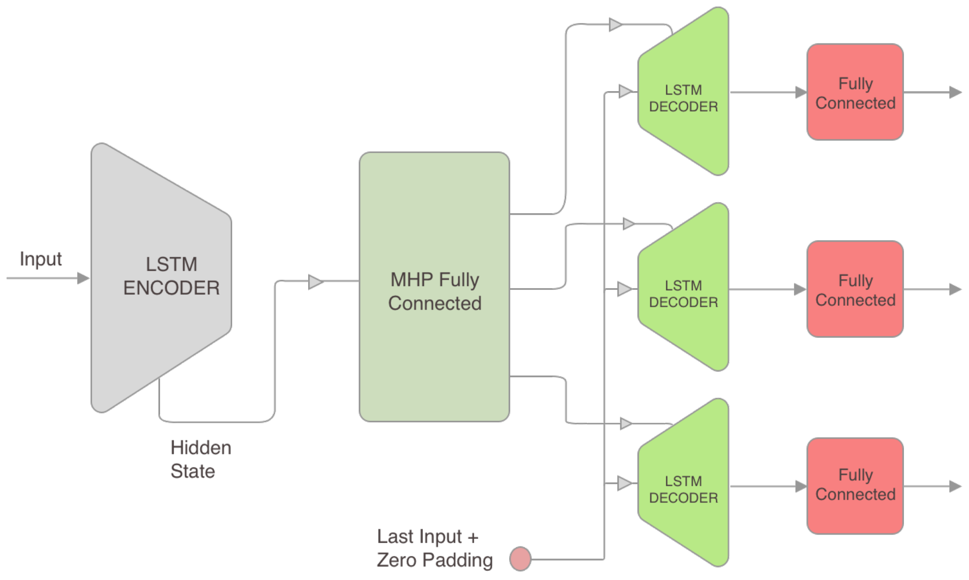


Figure 5.6: Model 2

In this model, the layer we use to generate multiple initial states must learn when an initial situation is ambiguous and consequently output different representations; these will be used by the decoder to predict different trajectories.

Both the models have an encoder-decoder architecture and predict as an output a vector of trajectories $\tilde{\mathbf{Y}}^n_t$. We force the predicted points to be consistent with the ground truth through the whole trajectory, employing the mean euclidean distance (L2-norm) as general loss.

$$\mathcal{L}_e(Y^n, \tilde{Y}^n) = \sum_{t=0}^{T_{pred}} ||y_t^n - \tilde{y}_t^n||_2 \tag{5.1}$$

The final step is to embed our loss into the MHP meta-loss function (Equation 4.8) so as to take into consideration the multiple hypothesis prediction.

$$\mathcal{M}(Y^n, \tilde{\mathbf{Y}}^n) = \sum_{k=1}^{K} \hat{\delta}(k = c)\mathcal{L}_e(Y^n, \tilde{Y}_k^n) \tag{5.2}$$

$$c = arg\min_h \mathcal{L}_e(Y^n, \tilde{Y}_h^n) \tag{5.3}$$

Minimising this function yields a Voronoi tessellation of the output space, as described in Chapter 4.

5.3 Multiple Sequence Classification

We also want to try MHP on a sequence classification problem. To do so we exploit the architecture used by Scheel et. al [49]. In their work, they consider the problem of assessing a situation with respect to the lane changing problem. Basically they output if a car is about to change lane (left or right) in every time-step of a sequence. They outperform existing methods on a public available dataset.

For this reason we decided to use the same dataset preprocessing that collects the car's velocity, lateral velocity, distance from the middle of the lane, lane position (number), information about the neighbouring vehicles and distance from previous cars.

The simple LSTM model takes as input an embedding of the previous information (or a subset of it), passes the result through a fully connected layer to reduce the dimensions and finally use a softmax layer to produce the probabilities. We use the same model, but we replicate the fully connected layer in order to have more than one hypothesis.

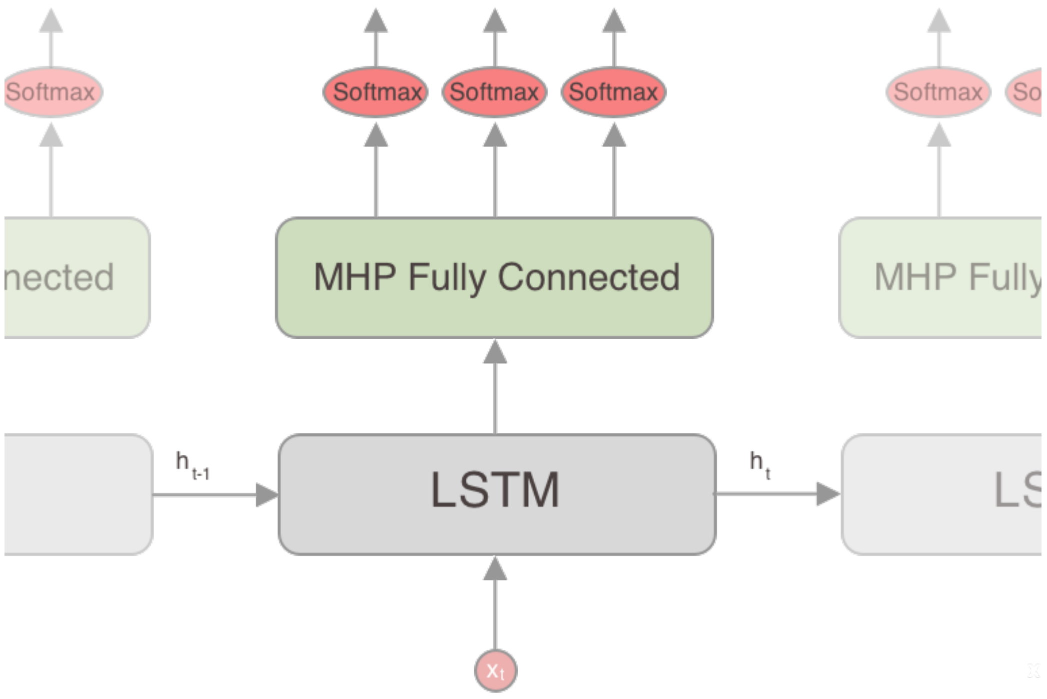


Figure 5.7: Classification model

We also embed the standard model loss, which is cross-entropy over the trajectory, into the MHP meta-loss as already explained in Equations 5.2 and 5.3

Chapter 6

Experiments

In this chapter we report both the quantitative and qualitative results that we obtained testing our models.

In Section 6.1 we present the experimental results on the trajectory prediction models. We confront our MHP model with the single hypothesis one and the state-of-the-art [TODO- but explain differences!!]; we also show some ambiguous situation where the models explore the different possibilities.

In Section 6.2 we compare accuracy, precision and recall, with the help of a particular metric, of the multiple classification model with respect to the standard one.

6.1 Trajectory Prediction

In this section we evaluate both our MHP models on a synthetic dataset and two publicly available datasets: Stanford Drone Dataset [47] and ETH [42].

6.1.1 Synthetic Dataset

This dataset is the one we introduced in Section 5.1. Examples of the trajectories are showed in Figure 5.1. We mainly used this dataset to derive the two models presented in Section 5.2 and to refine the hyperparameters. On such a simple dataset the differences between the single-prediction model and the MHP ones are minimal, also the two different MHP models return very similar trajectories. We therefore show only qualitative results obtained with

Model 2 (Figure 5.6). Nonetheless, the experiments carried out convinced us to continue our research and test our model on real examples.

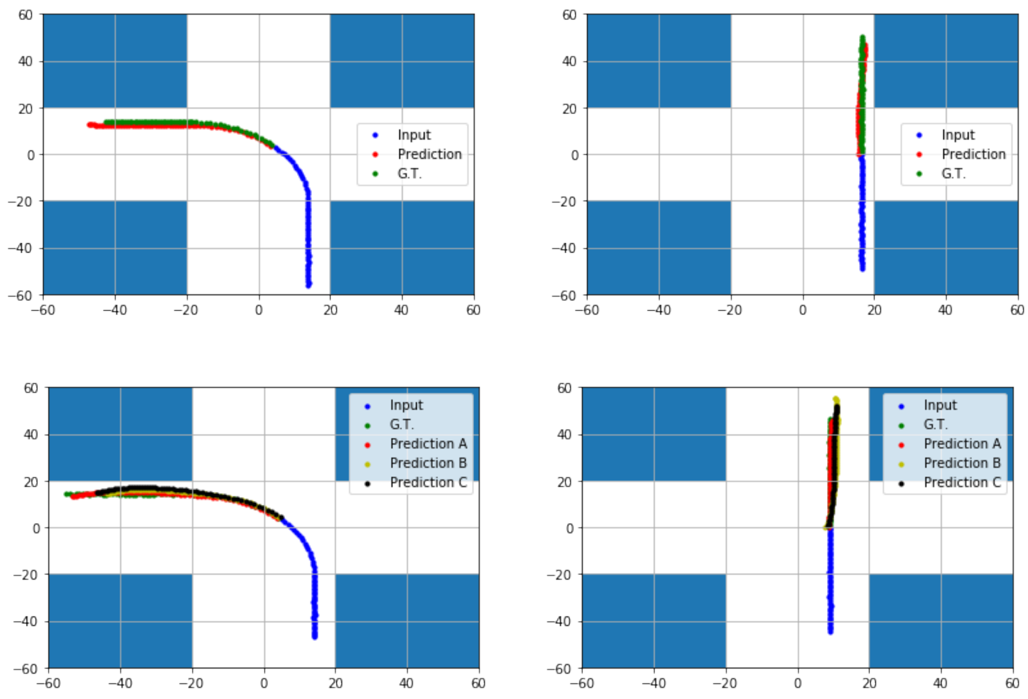


Figure 6.1: Differences between single hypothesis and multiple hypotheses in unambiguous situations

As we can see in Figure 6.1 the standard model and the MHP model work fine. The situations are obviously non-ambiguous therefore the predicted trajectories are very close to the ground truth. Also we can see how all the hypotheses predicted by our model are similar, showing that there is no uncertainty in the result.

In an ambiguous situation (Figure 6.2), however, we can see how the single prediction model is not able to output a feasible trajectory. Our model, instead, shows three the very different outcomes, highlighting also the fact that with the given input is not possible to return a certain answer.

6.1.2 Implementation Details

In the next experiments we use the two our models (MHP1, MHP2) and a single-prediction model (SHP) with an encoder-decoder architecture (basi-

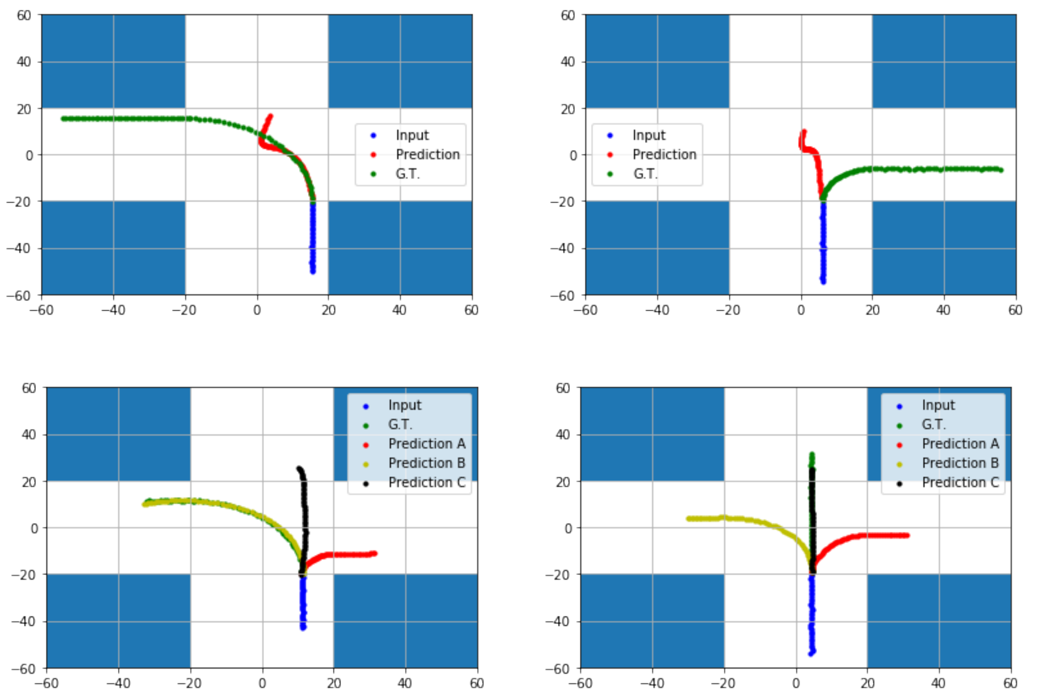


Figure 6.2: Differences between single hypothesis and multiple hypotheses in ambiguous situations

cally Model 1 with a single hypothesis). SHP loss is defined as the mean L2-norm (Equation 5.1). All the LSTMs, both encoders and decoders, have a fixed hidden state dimension of 64. We set the MHP ϵ equal to 0.15, higher than the one used in [48], since with 0.05 the networks are not able to learn correctly the different possibilities. We train with ADAM (learning rate 0.0001) on 200 epochs with an early stop patience of 10. We also add a regularisation term to prevent overfitting. All the models are implemented in Tensorflow and trained on a single low-performance GPU.

We report the errors with two metrics, similarly to [4] and [22], that in the MHP case are computed on the *best hypothesis*:

- *Average Displacement Error* - ADE: the average L2-norm of the difference between the predicted trajectory points and the respective ground truth.

- *Final Displacement Error* - FDE: the L2-distance between the predicted last points and ground truth last point.

We compare ADE and FDE of the following models:

- Single Hypothesis Model SHP.
- Model 1: with three (MHP1-3), five (MHP1-5) and seven (MHP1-7) hypotheses predicted.
- Model 2: with three (MHP2-3), five (MHP2-5) and seven (MHP2-7) hypotheses predicted.

6.1.3 ETH Dataset

The ETH Dataset [42] is composed of two scenes (ETH and Hotel) each containing 750 pedestrians. These dataset represent crowded scenarios, in fact it has been previously used in order to test social modelling architectures. The trajectories represent a pedestrian position every 0.4s. We test our models trying to predict 12 time-steps (4.2s). In these experiments the errors are in meters.

Model	Loss	ADE	FDE
SHP	0.077	0.76	1.15
MHP1-3	0.075	0.87	1.28
MHP1-5	0.078	0.82	1.24
MHP1-7	0.078	0.99	1.44
MHP2-3	0.078	0.61	0.91
MHP2-5	0.093	0.82	1.38
MHP2-7	0.067	0.82	1.29

Table 6.1: Quantitative Results on ETH (lower is better)

The quantitative results show that our models are able to predict at least as good as the single prediction model. In Table 6.1 we can see that the MHP2-3 best hypothesis is better than SHP. In Hotel we do not outperform the SHP model but the result are still promising. Also we can notice that

Model	Loss	ADE	FDE
SHP	0.093	0.41	0.54
MHP1-3	0.95	0.50	0.66
MHP1-5	0.128	1.10	1.58
MHP1-7	0.128	1.29	1.88
MHP2-3	0.124	0.68	0.94
MHP2-5	0.124	0.82	1.22
MHP2-7	0.123	0.84	1.10

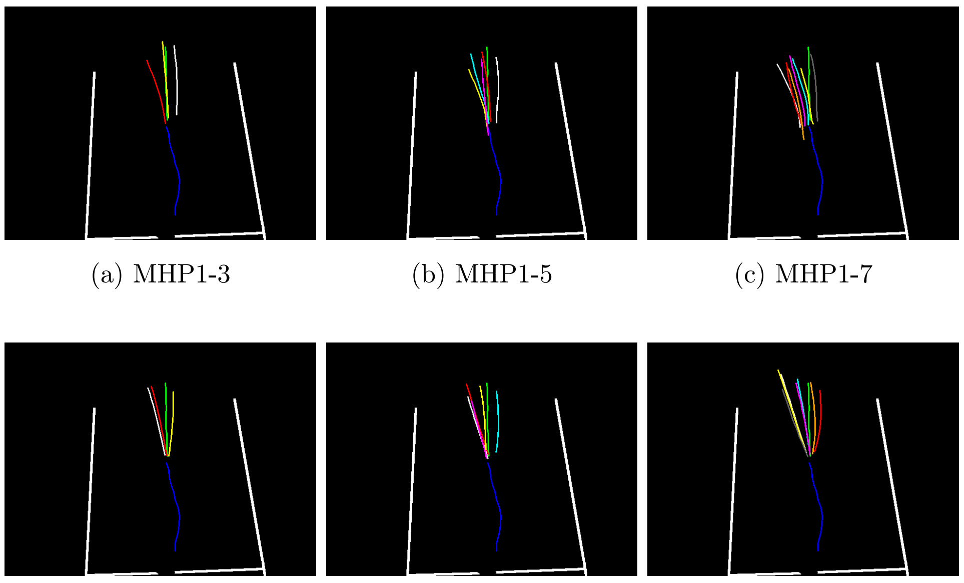
Table 6.2: Quantitative Results on Hotel (lower is better)

the errors are higher with more hypotheses, this may denote that the right number of hypothesis is three or that the model is not able to produce more than three trajectories.

To outperform the SHP model, however, is not the goal of this work. We want to test if the model is able to predict multiple feasible outcomes. This can happen only in a small subset of the whole dataset, the ambiguous scenes, and therefore the MHP usefulness is not clearly shown by the numbers. For this reason we now present some visual results.

In Figure 6.3 we can see how the models predict the different trajectories in one example of the ETH scene. Model 2, in particular, is able to generate always feasible trajectories looking at the starting point and the direction. Model 1, instead, provide trajectories that are a bit too far from the input endpoint; some of them are completely unfeasible. For instance in Figure 6.3b we can already see some trajectories that start from prohibitive positions (pink, white). The same happens in Figure 6.3c, in this case the best trajectory (grey) with respect to the ground truth is very unlikely to be feasible. All the trajectories predicted by Model 2 are instead interesting, some of them resemble the ground truth, others point towards another, but still real-looking, direction.

In Figure 6.4 we can see some visual results on the Hotel scene. Qualitatively we can say what we already noticed on ETH, Model 2 tends to give more plausible results, even if the errors are lower for Model 1. In fact, al-


(a) MHP1-3 (b) MHP1-5 (c) MHP1-7

(d) MHP2-3 (e) MHP2-5 (f) MHP2-7

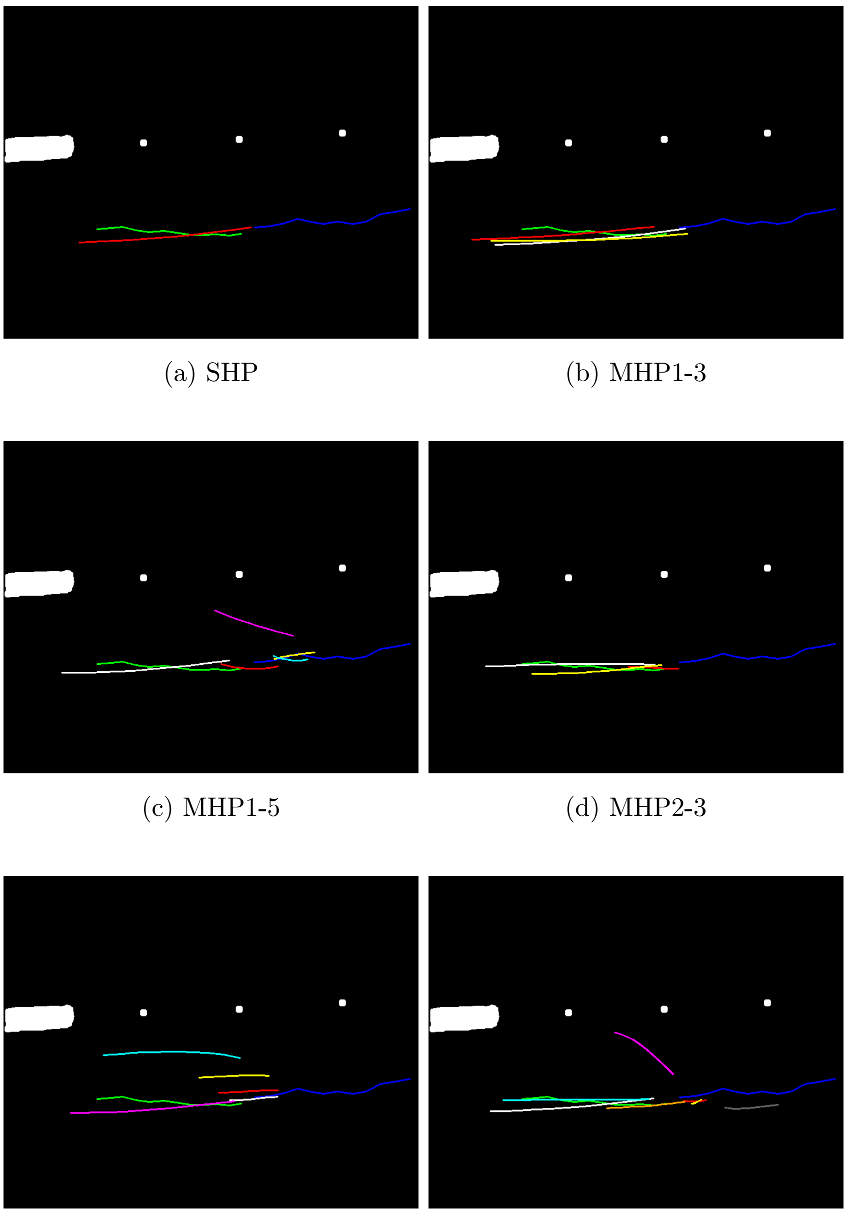
Figure 6.3: Visual results on ETH dataset (Blue=Input, Green=Ground Truth, other=Prediction

ready with five hypotheses (Figure 6.4), Model 1 outputs three sequences that are highly unlikely. We can also notice that every model is alway predicting at least one hypothesis that is very close to the real one. This indicates that the others represent tentatives to explore the possibile directions, given the examples analysed in the training set.

In this scene however the network seems more confused, one possible answer is that the number of example is lower and the trajectories do not follow certain predetermined paths (like ETH). Moreover, the trajectories in this scene are extremely linear so the model may have difficulties in generating different, but feasible, predictions.

6.1.4 Stanford Drone Dataset

This dataset contains aerial videos of pedestrians, cars, bikers and skaters captured in the Stanford University campus. The dataset is divided in different scenes and for each scene there are several videos. Since the trajectories in ETH dataset are mostly linear, we use a scene called "deathCircle" that



(a) SHP (b) MHP1-3

(c) MHP1-5 (d) MHP2-3

(e) MHP2-5 (f) MHP2-7

Figure 6.4: Visual results on Hotel dataset (Blue=Input, Green=Ground Truth, other=Prediction

contains trajectories of agents approaching a roundabout. These trajectories are not linear and most of them follow the roads or the sidewalks, we can therefore feed the network with sequences that are ideally following some rules. We can test if the models are able to generate the different outcomes given an ambiguous input sequence. All the positions are given as the four corner of a bounding box, we compute the actual position as the center of the box. The resulting trajectories may be therefore segmented in some points.

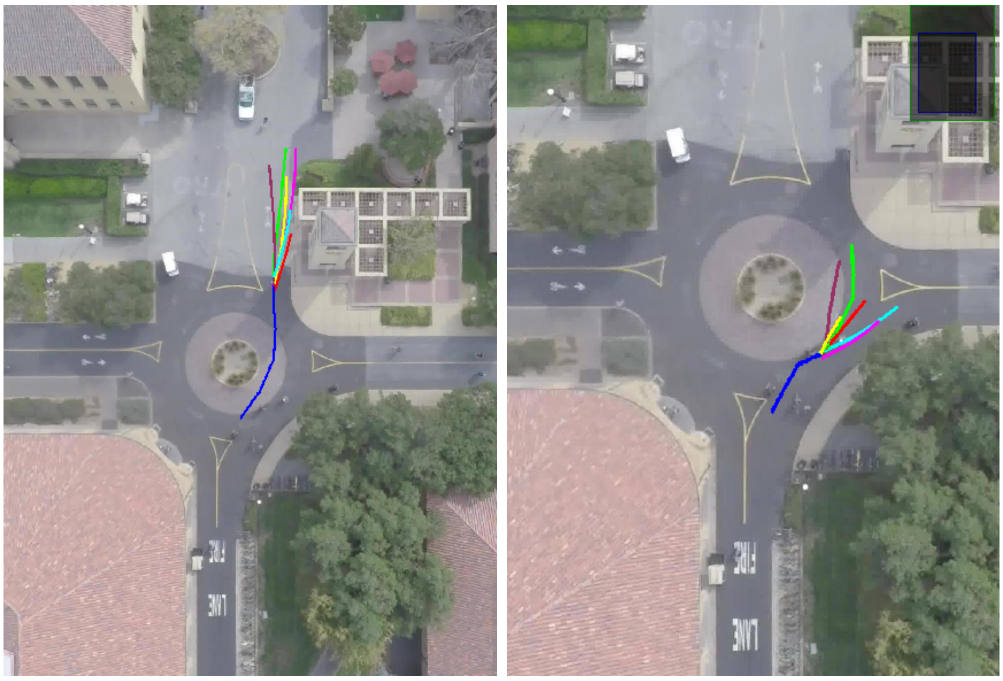
We first start with a quantitative evaluation. Since there is not a conversion between real world coordinate and pixels, ADE and FDE are shown in pixels. As we can see in Table 6.3 both our models with five and three hypotheses outperform the SHP model. Using the meta-loss we are able to reach better results on the best hypothesis. On this dataset, as well as in ETH the models predicting more hypotheses have worse performance.

With this dataset we can finally show that the model learns to predict feasible trajectories in ambiguous situation. As we show in Figure 6.5, when the input sequence represent a non ambiguous situation all the hypotheses converge and are similar to the ground truth. In fact in the first image a car exits the roundabout and there are no viable options but going straight. In the image (b), instead, the car is entering the road circle and the predicted trajectories split between going right or straight (and maybe eventually left).

Model	Loss	ADE	FDE
SHP	0.109	68.45	137.60
MHP1-3	0.087	63.93	124.52
MHP1-5	0.075	63.45	127.31
MHP1-7	0.089	96.47	178.14
MHP2-3	0.086	66.66	129.57
MHP2-5	0.075	67.15	134.36
MHP2-7	0.087	79.81	150.93

Table 6.3: Quantitative Results on Stanford deathCircle video1

In Figure 6.6 we see two other interesting examples. In image (a) we can see an input sequence of, presumably, a pedestrian crossing the road. In this case the network learned how to model also pedestrian trajectories in the sidewalk. In image (b), instead, we show an example in which the network trajectories go from the road to the sidewalk. This last result is still



(a) Non Ambiguous (b) Ambiguous

Figure 6.5: Difference between the outputs in ambiguous an non ambiguous situations

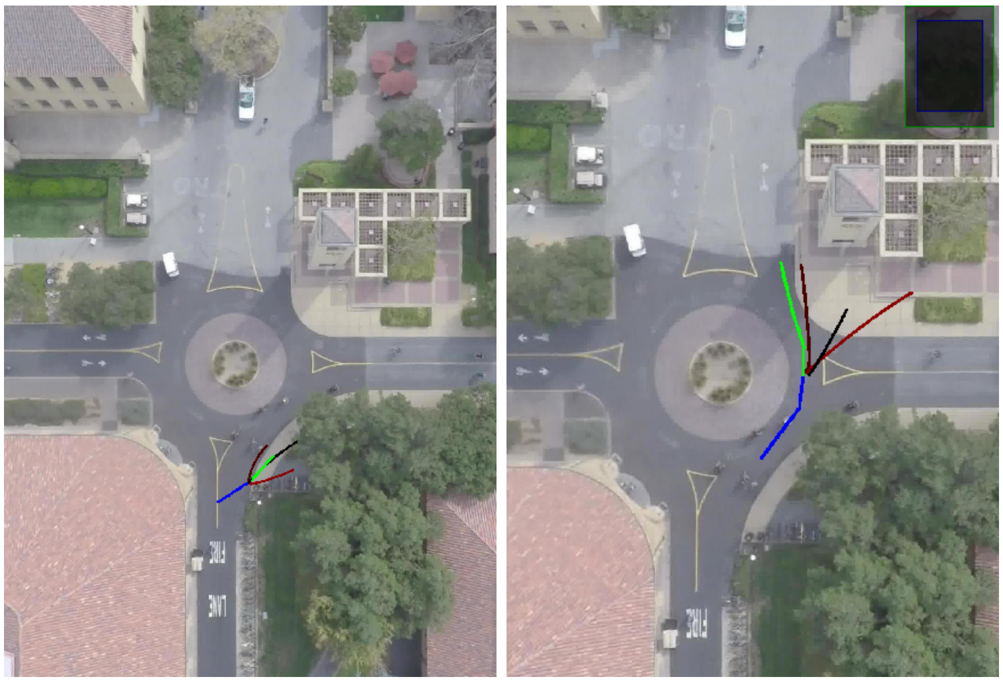
semi-plausible since the examples are mixed because, for instance, bikers ride both on road and sidewalk.

6.2 Sequence Classification

For the sequence classification problem we test on the same dataset as Scheel et al. [49], the Next Generation Simulation(NGSIM) [39], that is composed of examples from Interstate 80 Freeway Dataset (I-80) and US Highway 101 Dataset (US-101). The datasets contains the vehicles trajectories as well as other information, such as the lane boundaries.

We compare an SHP model against our MHP model with three and five hypotheses. We train both the models with two different sets of features:

- Set 1: Lateral velocity and distance from the middle of the lane
- Set 2: Lateral velocity, distance from the middle of the lane, heading angle (direction), lateral acceleration, longitudinal velocity, temporal



(a) (b)

Figure 6.6: Visual examples of Multiple predicitons

distance from the six closest neighbouring vehicles (2 on each lane) and lane number

We define a true prediction (*tp*) when the model predicts a lane change correctly, a false prediction (*fp*) when a lane change is predicted but the driver change the lane in the other direction (e.g. predicted left change but it is right change), a false positive prediction (*fpp*) when we predict a manoeuvre that will not happen and a missed prediction as *mp*. In order to compare the models we use three metrics:

- **Accuracy**: the overall accuracy over the sequences.
- **Precision**: $\frac{tp}{tp+fp+fpp}$
- **Recall**: $\frac{tp}{tp+fp+mp}$

These metric have been introduced by Jain et al. [28] that, in their work, use RNNs to fuse multiple sensors streams and anticipate car drivers

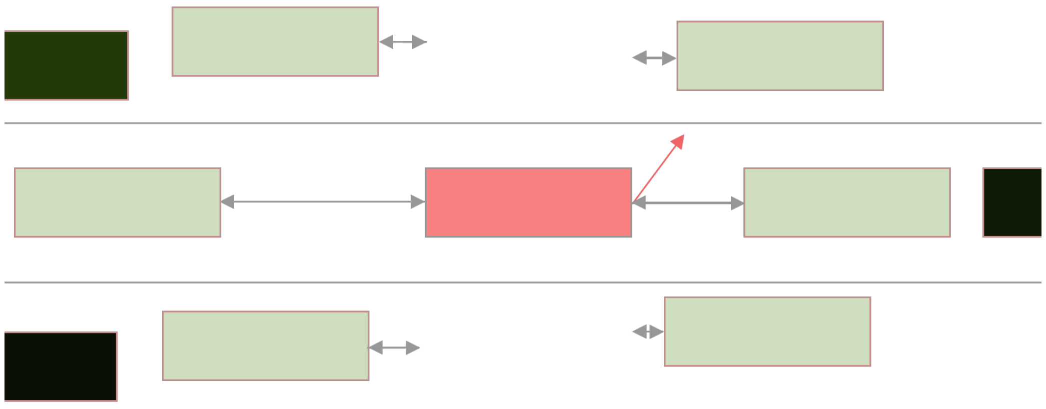


Figure 6.7: NGSIM scene, in red the considered car, in grey the neighbours and respective temporal distances

manoeuvres (lane changes, turns). Precision indicates how many of the predicted manoeuvres are correct while recall shows the amount of manoeuvres that are correctly classified. Notice that there is no distinction between a right lane change and a left lane change in this case.

A prediction is considered positive in case it is above a certain threshold. The threshold is a float number between 0 and 1 in case of SHP model. In order to define a positive prediction for MHP model it is necessary that more than a fixed threshold T hypotheses agree on the same prediction, therefore T must be an integer value between 1 and the number of hypotheses. We also consider the accuracy (on the best hypothesis in MHP models) but, since the lane changes are very rare, correctly classifying a lane change more important than the rest of the sequence.

If we consider the accuracy, our MHP models outperform the SHP with both features set. We could not obtain a model that is better in both precision and recall at the same time but we still have the peak values depending on which threshold we choose. As expected, if T is low recall is higher but the model outputs many false positives. On the other hand, a high threshold yields to very precise prediction, increasing the number of missed manoeuvres.

Model	Accuracy	Threshold	Precision	Recall
SHP	63%	0.8	55%	90%
		0.9	71%	87%
MHP3	66%	2	45%	90%
		3	88%	75%
MHP5	77%	3	51%	88%
		4	61%	74%
		5	94%	63%

Table 6.4: Sequence Classification with Set 1 features

Model	Accuracy	Threshold	Precision	Recall
SHP	63%	0.8	49%	87%
		0.9	68%	83%
MHP3	71%	2	47%	88%
		3	91%	66%
MHP5	70%	3	53%	77%
		4	55%	76%
		5	95%	62%

Table 6.5: Sequence Classification with Set 2 features

Chapter 7

Conclusion and Future Work

In this thesis we first describe how the multiple hypothesis predictions models, proposed by Rupprecht et al. [48], work and how a feedforward model can be modified to predict more than one feasible output.

Then, we exploit this framework to convert recurrent architectures into their respective MHP version in order to generate or classify time series. We defined two different models for regression problems that can easily retrofit single-prediction models with an encoder-decoder architecture. We tested these two models on public datasets and showed their ability to learn as good as the single hypothesis models or better in many cases. We show how the MHP models outperform the SHP one in ambiguous situations, where they are able to generate different feasible predictions.

We also test the MHP in a classification problem showing how predicting multiple classes increases the accuracy with respect to SHP. MHP models are also able to reach the highest values of precision and recall in this particular problem.

In the future we want to introduce social behaviour modelling and scene understanding in the trajectory prediction models to compare the results with state of the art papers, that already implement that. We also want to test the MHP classification with more complex models, like the ones we see in related work.

Bibliography

[1] By balu ertl [cc by-sa 1.0 (https://creativecommons.org/licenses/by-sa/1.0)], from wikimedia commons. 26

[2] By balu ertl [cc by-sa 4.0 (https://creativecommons.org/licenses/by-sa/4.0)], from wikimedia commons. 26

[3] Hirotugu Akaike. Fitting autoregressive models for prediction. 21:243–247, 1969. 21

[4] Alexandre Alahi, Kratarth Goel, Vignesh Ramanathan, Alexandre Robicquet, Li Fei-Fei, and Silvio Savarese. Social lstm: Human trajectory prediction in crowded spaces. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 22, 35, 43

[5] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. 2014. 21

[6] Tamer Basar. *A New Approach to Linear Filtering and Prediction Problems*. IEEE, 2001. 21

[7] Christopher M. Bishop. Mixture density networks. Technical report, 1994. 22

[8] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001. 8

[9] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. 2014. 21, 22

[10] Jan Chorowski, Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. End-to-end continuous speech recognition using attention-based recurrent NN: first results. 2014. 21

[11] Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. 2014. 22

[12] Junyoung Chung, Kyle Kastner, Laurent Dinh, Kratarth Goel, Aaron C. Courville, and Yoshua Bengio. A recurrent latent variable model for sequential data. 2015. 21

[13] Debadeepta Dey, Varun Ramakrishna, Martial Hebert, and J. Andrew Bagnell. Predicting multiple structured visual interpretations. *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 2947–2955, 2015. 23

[14] Jeff Donahue, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, Kate Saenko, and Trevor Darrell. Long-term recurrent convolutional networks for visual recognition and description. 2014. 21

[15] Qiang Du, Vance Faber, and Max Gunzburger. Centroidal voronoi tessellations: Applications and algorithms. *SIAM Rev.*, 41(4):637–676, 1999. 28

[16] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 12:2121–2159, 2011. 15

[17] Katerina Fragkiadaki, Sergey Levine, and Jitendra Malik. Recurrent network models for kinematic tracking. 2015. 21

[18] X. Geng, C. Yin, and Z. Zhou. Facial age estimation by learning from label distributions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(10):2401–2412, 2013. 23

[19] Alex Graves. Generating sequences with recurrent neural networks. 2013. 21

[20] Alex Graves and Navdeep Jaitly. Towards end-to-end speech recognition with recurrent neural networks. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML, pages 1764–1772, 2014. 21

[21] Alex Graves, Abdel-rahman Mohamed, and Geoffrey E. Hinton. Speech recognition with deep recurrent neural networks. 2013. 21

[22] Agrim Gupta, Justin Johnson, Li Fei-Fei, Silvio Savarese, and Alexandre Alahi. Social GAN: socially acceptable trajectories with generative adversarial networks. 2018. 22, 35, 43

[23] Abner Guzmán-rivera, Dhruv Batra, and Pushmeet Kohli. Multiple choice learning: Learning to produce multiple structured outputs. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1799–1807. Curran Associates, Inc., 2012. 23

[24] M. A. Hearst, S. T. Dumais, E. Osuna, J. Platt, and B. Scholkopf. Support vector machines. *IEEE Intelligent Systems and their Applications*, 13(4):18–28, 1998. 8

[25] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, 1997. 18

[26] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Netw.*, 4(2):251–257, 1991. 11

[27] Marcus Hutter. The human knowledge compression contest. 2012. 21

[28] Ashesh Jain, Hema Swetha Koppula, Shane Soh, Bharad Raghavan, Avi Singh, and Ashutosh Saxena. Brain4cars: Car that knows before you do via sensory-fusion deep learning architecture. 2016. 22, 50

[29] Andrej Karpathy, Armand Joulin, and Fei-Fei Li. Deep fragment embeddings for bidirectional image sentence mapping. 2014. 21

[30] Andrej Karpathy and Fei-Fei Li. Deep visual-semantic alignments for generating image descriptions. 2014. 21

[31] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. 2014. 15

[32] A.N. Kolmogorov. *Foundations of the theory of probability, pages 47-64.* Chelsea Pub. Co., 1950. 26

[33] Kimin Lee, Changho Hwang, KyoungSoo Park, and Jinwoo Shin. Confident multiple choice learning. 2017. 23

[34] Namhoon Lee, Wongun Choi, Paul Vernaza, Christopher Bongsoo Choy, Philip H. S. Torr, and Manmohan Krishna Chandraker. DESIRE: distant future prediction in dynamic scenes with interacting agents. 2017. 22

[35] Stefan Lee, Senthil Purushwalkam, Michael Cogswell, Viresh Ranjan, David J. Crandall, and Dhruv Batra. Stochastic multiple choice learning for training diverse deep ensembles. 2016. 23

[36] Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. Building a large annotated corpus of english: The penn treebank. *Comput. Linguist.*, 19(2):313–330, 1993. 21

[37] P. McCullagh and J.A. Nelder. *Generalized Linear Models, Volume 37.* CRC Press, 1989. 21

[38] Joe Yue-Hei Ng, Matthew J. Hausknecht, Sudheendra Vijayanarasimhan, Oriol Vinyals, Rajat Monga, and George Toderici. Beyond short snippets: Deep networks for video classification. 2015. 21

[39] NGSIM. “ngsimproject”, https://ops.fhwa.dot.gov/trafficanalysistools/ngsim.htm. 49

[40] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. Understanding the exploding gradient problem. 2012. 18

[41] Dario Pavllo, David Grangier, and Michael Auli. Quaternet: A quaternion-based recurrent model for human motion. 2018. 21

[42] S. Pellegrini, A. Ess, K. Schindler, and L. van Gool. You'll never walk alone: Modeling social behavior for multi-target tracking. In *2009 IEEE 12th International Conference on Computer Vision*, pages 261–268, 2009. 41, 44

[43] M.B. Priestley. *Spectral Analysis and Time Series: Multivariate series, prediction and control*. Probability and Mathematical Statistics. Academic Press, 1981. 21

[44] Joaquin Quiñonero Candela and Carl Edward Rasmussen. A unifying view of sparse approximate gaussian process regression. *J. Mach. Learn. Res.*, 6:1939–1959, 2005. 21

[45] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986. 8

[46] Carl Edward Rasmussen. *Gaussian Processes in Machine Learning*, pages 63–71. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004. 21

[47] Alexandre Robicquet, Amir Sadeghian, Alexandre Alahi, and Silvio Savarese. Learning social etiquette: Human trajectory understanding in crowded scenes. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *ECCV*, pages 549–565, Cham, 2016. Springer International Publishing. 41

[48] Christian Rupprecht, Iro Laina, Maximilian Baust, Federico Tombari, Gregory D. Hager, and Nassir Navab. Learning in an uncertain world: Representing ambiguity through multiple hypotheses. 2016. 6, 23, 25, 28, 43, 53

[49] Oliver Scheel, Loren Arthur Schwarz, Nassir Navab, and Federico Tombari. Situation assessment for planning lane changes: Combining recurrent models and prediction. 2018. 22, 38, 49

[50] D. Tran, W. Sheng, L. Liu, and M. Liu. A hidden markov model based driver intention prediction system. In *2015 IEEE International Conference on Cyber Technology in Automation, Control, and Intelligent Systems (CYBER)*, pages 115–120, 2015. 22

[51] J. M. Wang, D. J. Fleet, and A. Hertzmann. Gaussian process dynamical models for human motion. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(2):283–298, 2008. 21

[52] Christopher K. I. Williams. *Prediction with Gaussian processes: from linear regression to linear prediction and beyond*, pages 599–621. MIT, 1999. 21

[53] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron C. Courville, Ruslan Salakhutdinov, Richard S. Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. 2015. 21

Ringraziamenti

Un ringraziamento speciale ai miei genitori e ai miei nonni che mi hanno permesso di raggiungere questo traguardo. Alla mia ragazza, Chiara, per avermi supportato (ma soprattutto sopportato) fino ad oggi. Agli amici di Romagna Ago, Adam, Gigi, Mazzo e Otto per le serate, le feste e le trasferte. Gianna, Buro, Fucio, Simo per aver condiviso esami e fatiche. Gabri, Lori e Ricky per ogni progetto preparato assieme. Bombo per esserci sempre stato negli ultimi cinque anni, per ogni esame preparato assieme, per ogni volta che siamo tornati a tarda notte e mi hai fatto dormire sul divano e per aver contribuito a rendere Bologna la mia seconda casa. Grazie a tutti quanti (anche quelli che non ho citato) per aver reso questi anni fantastici. Ringraziamento aggiuntivo a Bombo e Lori per avermi ospitato un mese a testa mentre non avevo un appartamento.
I would like to thank Prof. Di Stefano and Dr. Tombari for giving me the opportunity to prepare this thesis at the TU München, Oliver for all the ideas and the time spent helping and all the people from CAMP chair. Also a big thank you to all the new (and old) friends from München: Alessio, Alessandro, Steve, Michael and Ted. Last but not least: Shounak, Sam and Pouya the crazy people i needed in Germany.
Special thanks to DESTRAGE for the music that I listened throughout these years.

> "You think you learned the rules, but instead you just narrowed your views. Learn on your skin, not from those books. Try again, don't be afraid. Fail as much as you can. Who cares in the end?"
>
> –*Destrage, "Destroy Create Transform Sublimate"*