ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Broad Discourse Context for Language Modeling

Master Thesis

Moisés Torres

November 1, 2017

Supervisor:
Prof. Dr. Thomas Hofmann

Co-supervisors:
Florian Schmidt
Paulina Grnarova

Department of Computer Science, ETH Zürich

Abstract

Discourse understanding summarizes a speaker's ability to perform multi-sentence reasoning. This in turn requires temporal reasoning, resolving co-reference chains, identifying named entities and keeping track of the state of a conversation. In language modeling, the de-facto standard task for text generation systems, deep neural network have become the state of the art. Attempts to transfer similar architectures to a dialogue task directly, have revealed the lack of a rigorous evaluation metric

As a middle ground, newly released datasets try to cast existing NLP problems in a discourse context. For example, collects hard word-prediction tasks to put to the test genuine understanding of language models. To succeed on this task, a language model cannot simply rely on local context, but must be able to keep track of information in the broader discourse. However, the established evaluation methodologies from language modeling still apply.

Previous work like have shown that plain RNNs' memory representation may not be enough to effectively capture long term dependencies. Thus, several approaches have been proposed to tackle this problem: attention , memory networks or latent variable RNNs, among others. In this thesis we implement several baselines of enhanced language models using some of the aforementioned techniques and evaluate them on hard-word prediction tasks like. Based on our empirical findings, we propose and compare extensions to tackle the identified shortcomings.

First paragraph, high level description of the problem. Check project proposal!
- Describe thesis approach, mention lambada maybe
- What focus for the abstact? Still focus on discourse?

Acknowledgments

I would like to thank...

Finish acknowledgements

Contents

Chapter 1

Introduction

Here comes the intro...

Finish intro (3 pages)

1.1 Problem Statement and Motivation

motivation...

Finish motivation (1 page)

1.2 Thesis Contributions

contributions...

Finish contributions (1 page)

1.3 Thesis Outline

outline...

Finish outline (1 page)

Chapter 2

Related Work

This chapter will describe [1] the state-of-the-art [2] of...

Finish related work (3 pages)

Chapter 3

Neural Language Modeling

In this chapter, we introduce the notation used throughout the thesis and give a brief overview of the development of neural language modeling (NLM) since its inception. We also review some of the main weaknesses shown by this family of models and how they have been addressed in the literature.

- Finish this (7-8 pages)
- Mention CNN LMs ?
- Softmax variants (hierarchical)?
- Modern models (highway)?
- Include references for n-grams?
- Figures?

3.1 Notation

Before continuing, we will define the notation used in this thesis:

- Scalars are denoted with lowercase letters, such as x.
- Vectors (of size N) are denoted with bold lowercase letters, such as $\mathbf{x}$ with its i-th element $\mathbf{x}_i$, and are always assumed to be column vectors.
- Matrices (of size $N \times M$) are denoted with uppercase letters, such as X with X_{ij} as its (i, j)-th element.

3.2 Background

Prior to introducing the specifics of NLMs, we will formalize the task at hand and introduce some of its core concepts.

3.2.1 Language Modeling

First, we define a **word-based language model** as a model able to compute the probability of a sentence or sequence words $P(w_1, \ldots, w_n)$. Such models are of great use in tasks where we have to recognize words in noisy or ambiguous input such as speech recognition or machine translation, among others.

If now we decompose the joint probability of a sequence using the chain rule of probability as shown in Equation 3.1, we observe that the function that needs to be estimated boils down to the conditional probability of a word given the history of previous words. However, taking into account the whole context poses a problem as language is creative and any particular sequence might have occurred few (or no) times before. Many of the models that we will introduce opt to approximate the real conditional distribution by making a Markov assumption as shown in Equation 3.2. This means that the probability of an upcoming word is fully characterized by the $n-1$ previous ones. Despite seeming an incorrect assumption for a complex source of information such as language, it has been proven to work really well in practice.

$$\begin{aligned} p(w_1, \ldots, w_n) &= p(w_1)p(w_2|w_1)p(w_3|w_1^2)\ldots p(w_n|w_1^{n-1}) \\ &= \prod_{k=1}^{n} p(w_k|w_1^{k-1}) \end{aligned} \tag{3.1}$$

$$p(w_k|w_1^{k-1}) \approx p(w_k|w_{k-1}^{k-n}) \tag{3.2}$$

3.2.2 Evaluation

Following a common practice in machine learning, we use a test set in order to evaluate our models. In the case of language modeling we have a word sequence $W_1^n = \{w_1, \ldots, w_n\}$ and the better the model is, the higher the probability it will assign to this sequence. Rather than working directly with raw probabilities we define a metric called **perplexity**, which is the geometric average of the inverse of the probability over the test set, as shown in Equation 3.3. Therefore, lower perplexity is better.

$$\begin{aligned} \text{Perplexity}(W_1^n) = p(W_1^n)^{-\frac{1}{n}} &= \sqrt[n]{\frac{1}{p(W_1^n)}} \\ &= \sqrt[n]{\frac{1}{\prod_{k=1}^{n} p(w_k|W_1^{k-1})}} \end{aligned} \tag{3.3}$$

Moreover, we can regard language as a source of information and apply the Information Theory toolbox to find a different (and equivalent) interpretation of perplexity. For that we need to introduce the basic concept of **entropy** (Equation 3.4 shows its formulation for discrete variables), which measures the expected uncertainty or "surprise" S of the value of a random variable X. Without going into details, it is easy to see that defining uncertainty as the negative logarithm (the specific base doesn't matter, but traditionally it is assumed to be 2) of the probability of each event matches our intuition (like $S(p) > S(q)$ then $p < q$).

$$H(X) = \mathbb{E}[S(X)] = -\sum_{x \in \mathcal{X}} p(x) \log_2(p(x)) \quad \text{with} \quad S(\cdot) = -\log_2(\cdot) \tag{3.4}$$

A difference when it comes to language is that it involves dealing with sequences W_1^n of discrete random variables. For a given language L we can define the entropy of a variable ranging over all possible sequences of length n. To obtain the entropy-per-word we would only need to normalize by n (Equation 3.5).

$$\frac{1}{n} H(W_1^n) = -\frac{1}{n} \sum_{W_1^n \in L} p(W_1^n) \log_2(p(W_1^n)) \tag{3.5}$$

However, in order to calculate the true entropy of a language we need to consider sequences of infinite length (Equation 3.6). Shannon-McMillan-Breiman theorem states that if the language is regular in certain ways we can take a single long enough sequence instead of summing over all possible sequences (* in Equation 3.6).

$$\begin{aligned} H(L) &= -\lim_{n \to \infty} \frac{1}{n} \sum_{W_1^n \in L} p(W_1^n) \log_2(p(W_1^n)) \\ &\overset{*}{=} -\lim_{n \to \infty} \frac{1}{n} \log_2(p(W_1^n)) \end{aligned} \tag{3.6}$$

Similarly we have **cross-entropy** which measures the relative entropy of p with respect to m, p being the true probability distribution and m a model (e.g. an approximation) of p. After applying Shannon-McMillan-Breiman theorem and assuming that n is large enough, we can see in Equation 3.7 the final formulation of the cross-entropy, which is used as the default loss function when optimizing neural language models.

$$\begin{aligned} H(P, M) &= -\lim_{n \to \infty} \frac{1}{n} \sum_{W_1^n \in L} p(W_1^n) \log_2(m(W_1^n)) \\ &\overset{*}{=} -\lim_{n \to \infty} \frac{1}{n} \log_2(m(W_1^n)) \approx -\frac{1}{n} \log_2(m(W_1^n)) \\ &= -\frac{1}{n} \sum_{k=1}^{n} \log_2(m(w_k | W_1^{k-1})) \end{aligned} \tag{3.7}$$

Finally, we can see in Equation 3.8 how cross-entropy and perplexity are connected. This relation gives raise to a nice interpretation of perplexity as branching factor: entropy measures uncertainty (in bits, if we use $\log_2$) but in

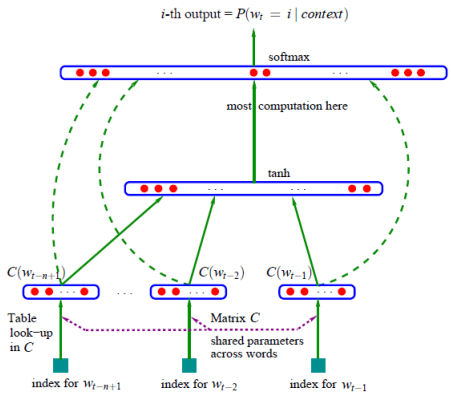
exponentiated form it's measured as the cardinality of a uniform distribution with equivalent uncertainty.

$$\text{Perplexity}(W_1^n) = 2^{\text{cross-entropy}} = 2^{H(p,m)} = m(W_1^n)^{-\frac{1}{n}} \tag{3.8}$$

3.3 Feed-Forward Neural Language Models (FFNLM)

Until the appearance of NLMs the most successful approaches were based on n-grams, which are Markov models that estimate words from a fixed window of previous words and estimate probabilities by counting in a corpus and normalizing. Due to their nature n-gram estimates intrinsically suffer from sparsity and several methods like smoothing, backoff and interpolation have been proposed to deal with this problem.

Along those lines, the first successful attempt of applying neural networks [3] raised the point that when modeling the joint distribution between many discrete random variables (such as words in a sentence), any change of these variables may have a drastic impact on the value of the estimated function. On the contrary, by using continuous variables we obtain better generalization because the function to be learned can be expected to have some local smoothness properties ("similar" words should get similar probabilities). While taking longer to train, this approach is able to achieve significantly better results by jointly learning word representations and a statistical language model.



$$\begin{aligned} \mathbf{x} &= [C(w_{t-1}), C(w_{t-2}), \ldots, C(w_{t-n+1})] \\ \mathbf{y} &= W\mathbf{x} + U\tanh(H\mathbf{x} + \mathbf{d}) + \mathbf{b} \\ \hat{\mathbf{p}}_i &= \hat{p}(w_t = i | w_{t-1}, \ldots, w_{t-n+1}) = \frac{e^{\mathbf{y}_i}}{\sum_n e^{\mathbf{y}_n}} \end{aligned} \tag{3.9}$$

Figure 3.1: Feed-Forward NLM architecture

Similar to n-grams, the model introduced in [3] conditions the probability of a word on the previous $n-1$ words. The main difference lies in the concept of "distributed feature vectors"; words are embedded into a vector-space by assigning them a continuous real-vector representation. As seen in

Figure 3.1, this is done via a look-up operation over the embedding matrix C. The concatenated word representations are then fed through one or more hidden layers and the resulting hidden representation is used to generate the unscaled log probabilities with a fully connected layer. Finally, a softmax operation produces a valid probability distribution over the full vocabulary.

3.4 Word Vectors

As we have seen in the previous section, distributed continuous vectors allow for "clever" smoothing by taking into account automatically learnt syntactic and semantic features. [4] picked up on this concept trying to find ways of training these vector representations more efficiently. The paper introduces a family of models known as **word2vec**, whose architecture matches the one from a FFNLM where the non-linear hidden layer has been removed (and we end up with a simple log bilinear model). The difference between them lies on the "fake" objective (fake in the sense that we are only interested in the resulting embeddings and not the actual outputs of the model) they optimize for learning the word embeddings:

- Continuous Bag-of-Words model (CBOW): given a symmetric window of size k around a specific position $\{w_{i-k}, \ldots, w_{i-1}, w_{i+1}, \ldots w_{i+k}\}$ we want to predict that word w_i. The term "bag-of-words" comes from the fact that the embeddings of the whole window are summed (instead of concatenated) and thus, order is not kept anymore.
- Continuous Skip-Gram model: given a specific position, we randomly sample words inside its surrounding window and try to predict them. Therefore, each training example is a tuple consisting of w_i as input and a word from the window as output.

In addition to a simplified architecture and a modified objective, further optimizations for the Skip-Gram model were introduced in the follow-up paper [5]. As we already saw in Equation 3.9??, most of the computation is done in the softmax operation over the full vocabulary. In order to avoid this, we cast our task to a binary classification problem by making use of a new objective called **negative sampling**. Inspired by noise contrastive estimation (NCE), the task is to distinguish the target word w_O from draws from a noise distribution $P_n(w)$ (e.g. unigram distribution) using logistic regression, where there are k negative samples for each data sample (Equation 3.10).

$$\mathcal{L}(\theta) = \log(P(w_O|w_I)) = \log(\sigma({\mathbf{v_{w_O}}}^\top \mathbf{v_{w_I}})) + \sum_{i=1}^{k} \log(-\sigma({\mathbf{v_{w_i}}}^\top \mathbf{v_{w_I}})) \quad \text{with} \quad w_i \sim P_n(w) \tag{3.10}$$

Another famous family of word vectors is **GloVe** [6], where the objective is a weighted (weighting function $f(\cdot)$) least squares fit of the log-counts (Equation 3.11). Rather than taking a predictive model approach to learn their vectors in order to improve their predictive ability (like word2vec), GloVe does dimensionality reduction on the co-occurrence counts matrix N.

$$\mathcal{L}(\theta, N) = \sum_{i,j:N_{ij}>0} f(N_{ij})(\log(N_{ij}) - (\mathbf{v_{w_O}}^\top \mathbf{v_{w_I}} + b_O + b_I))^2 \tag{3.11}$$

In summary, word vectors have become a standard in NLP and are used as input in all sorts of downstream applications such as sentiment analysis.

3.5 Recurrent Neural Language Models (RNLM)

So far all the models that we have seen (n-grams and FFNLM) have to explicitly use a fixed length context. Recurrent neural networks remove this limitation by introducing recurrent connections that allow information to cycle for an arbitrarily long time (although this is not true in practice). They learn to compress the whole history in low dimensional space (hidden state $\mathbf{h_t}$) that is sequentially updated by being blended with the current input (Equation 3.12). [7] was one of the first to successfully apply RNNs to language modeling.

$$\begin{aligned} \mathbf{h}(t) &= \sigma_h(W_h\mathbf{x}(t) + U_h\mathbf{h}(t-1) + \mathbf{b_h}) \\ \mathbf{y}(t) &= W_y\mathbf{h}(t) + \mathbf{b_y} \\ \hat{\mathbf{p}}(t)_i &= \hat{p}(w_t = i|\mathbf{h}(t)) = \frac{e^{\mathbf{y}(t)_i}}{\sum_n e^{\mathbf{y}(t)_n}} \end{aligned} \tag{3.12}$$

3.5.1 Backpropagation Through Time (BPTT)

Backpropagation is the standard training procedure used for neural networks. However when applied to the recurrent connections of an RNN, the gradients will depend on the previous timesteps (up to $t = 0$). Thus, we call Backpropagation Through Time (BPTT) to the application of backpropagation on an unrolled RNN, which accounts for this dependencies by summing up the gradients for each time step. In Equation 3.13 we see an example of this when calculating the gradient for U_h.

$$\frac{\partial \mathcal{L}(t)}{\partial U_h} = \frac{\partial \mathcal{L}(t)}{\partial \hat{\mathbf{p}}(t)} \frac{\partial \hat{\mathbf{p}}(t)}{\partial \mathbf{h}(t)} \frac{\partial \mathbf{h}(t)}{\partial U_h} = \sum_{k=0}^{t} \frac{\partial \mathcal{L}(t)}{\partial \hat{\mathbf{p}}(t)} \frac{\partial \hat{\mathbf{p}}(t)}{\partial \mathbf{h}(t)} \frac{\partial \mathbf{h}(t)}{\partial \mathbf{h}(k)} \frac{\partial \mathbf{h}(k)}{\partial U_h} \tag{3.13}$$

One of the main problems of BPTT is the high cost of a single parameter update, which makes it impossible to use for large numbers of iterations. **Truncated Backpropagation Through Time** (TBPTT), which is a modified version of BPTT, was introduced in [8] to work around this limitation. where the sequence is processed one timestep at a time, and every k_1 timesteps, it runs BPTT for k_2 timesteps, so a parameter update can be cheap if k_2 is small. Most implementations take $k_1 = k_2$.

3.5.2 Exploding and Vanishing Gradient

If input sequences are comprised of thousands of timesteps, then this will be the number of derivatives required for a single weight update. We can see this in Equation 3.14 by observing that $\frac{\partial \mathbf{h}(t)}{\partial \mathbf{h}(k)}$ is a chain-rule itself. Citing [9], *"a product of $t-k$ real numbers can shrink to zero or explode to infinity, so does this product of matrices"* and therefore, this can cause gradients to vanish or explode.

$$\frac{\partial \mathbf{h}(t)}{\partial \mathbf{h}(k)} = \prod_{t \geq i > k} \frac{\partial \mathbf{h}(i)}{\partial \mathbf{h}(i-1)} \tag{3.14}$$

For exploding gradients, clipping the norm of the gradients to a predefined threshold has proven to solve this problem. On the other hand, vanishing gradient translates into gradient contributions from "far away" steps becoming zero, and thus hindering the learning of long-range dependencies. The most popular solution for this problem has been the introduction of new cell architectures explicitly designed to deal with vanishing gradients such as Long Short-Term Memory (LSTM) [10] and Gated Recurrent Units (GRU) [11].

$$\begin{aligned}
\mathbf{f}(t) &= \sigma_g(W_f \mathbf{h}(t-1) + U_f \mathbf{x}(t) + \mathbf{b}_f) \\
\mathbf{i}(t) &= \sigma_g(W_i \mathbf{h}(t-1) + U_i \mathbf{x}(t) + \mathbf{b}_i) \\
\mathbf{o}(t) &= \sigma_g(W_o \mathbf{h}(t-1) + U_o \mathbf{x}(t) + \mathbf{b}_o) \\
\tilde{\mathbf{c}}(t) &= \sigma_c(W_c \mathbf{h}(t-1) + U_c \mathbf{x}(t) + \mathbf{b}_c) \\
\mathbf{c}(t) &= \mathbf{f}(t) \odot \mathbf{c}(t-1) + \mathbf{i}(t) \odot \tilde{\mathbf{c}}(t) \\
\mathbf{h}(t) &= \mathbf{o}(t) \odot \sigma_h(\mathbf{c}(t))
\end{aligned} \tag{3.15}$$

We will focus on the LSTM as the GRU cell is just a simplified version of the former. The main differences to a vanilla RNN are the introduction of a cell state $\mathbf{c}_t$ (that acts as internal memory) and gates $\mathbf{f}, \mathbf{i}, \mathbf{o}$. Gates are a way to optionally let information through and are learnt in such a way that the cell can remember long-range dependencies. As shown in Equation 3.15, the new cell state is formed by a combination of the previous one weighted by the

forget gate $\mathbf{f}(t)$ and the "newly proposed" state $\tilde{\mathbf{c}}(t)$ weighted by the input gate $\mathbf{i}(t)$.

To sum up, recurrent architectures using LSTM cells have become ubiquitous in a wide range of NLP tasks and are achieving state-of-the-art results in many of them.

Chapter 4

Rare Word Prediction

bla bla

Finish this

$$\mathcal{L} = -\sum_{j} \hat{y_{ij}} \log(p(y_{ij}|x_i)) - \log(g + \sum_{i \in I(y,x)} a_i) \tag{4.1}$$

Chapter 5

Experiments and Results

experiments...

Finish experiments and results

$$\mathcal{L} = -\sum_n \log(P(name|h_t)P(w_t|h_t, name) + (1 - P(name|h_t))P(w_t|h_t, notName)) \\ +\lambda(y_{name}\log(P(name|h_t)) + (1 - y_{name})\log(1 - P(name|h_t)) \tag{5.1}$$

		All		Names	
		Train	Dev	Train	Dev
Basic	Baseline	42.5	61.5	1000	130K
	Mixture ($\lambda = 100$)	55	69	4000	120K
Input dropout	Baseline	52.5	63	2000	140K
	Mixture ($\lambda = 100$)	65	72	8000	120K
State dropout	Baseline	48	62.5	1000	120K
	Mixture ($\lambda = 100$)	62.5	73	6000	120K
Output dropout	Baseline	62.5	65	5000	150K
	Mixture ($\lambda = 100$)	80	73	20K	80K
L2 regularization ($\beta = 0.01$)	Baseline	100	125	10K	400K
	Mixture ($\lambda = 100$)	107.5	119	8000	120K

Table 5.1: Perplexity

Chapter 6

Conclusion

brief remarks...

Finish conclusion (1/2 page)

6.1 Achieved Results

results...

Finish achieve results (1/2 page)

6.2 Future Work

future work...

Finish future work (1/2 page)

Bibliography

[1] Stephen Merity, Nitish Shirish Keskar, and Richard Socher. Regularizing and optimizing lstm language models. *arXiv preprint arXiv:1708.02182*, 2017.

[2] Edouard Grave, Armand Joulin, and Nicolas Usunier. Improving neural language models with a continuous cache. *arXiv preprint arXiv:1612.04426*, 2016.

[3] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155, 2003.

[4] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[5] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[6] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[7] Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Interspeech*, volume 2, page 3, 2010.

[8] Ilya Sutskever. Training recurrent neural networks. *University of Toronto, Toronto, Ont., Canada*, 2013.

[9] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, pages 1310–1318, 2013.

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

[11] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[12] Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Quan Ngoc Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The lambada dataset: Word prediction requiring a broad discourse context. *arXiv preprint arXiv:1606.06031*, 2016.

[13] Robin Jia and Percy Liang. Adversarial examples for evaluating reading comprehension systems. *arXiv preprint arXiv:1707.07328*, 2017.

[14] Michał Daniluk, Tim Rocktäschel, Johannes Welbl, and Sebastian Riedel. Frustratingly short attention spans in neural language modeling. *arXiv preprint arXiv:1702.04521*, 2017.

[15] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

[16] Caglar Gulcehre, Sungjin Ahn, Ramesh Nallapati, Bowen Zhou, and Yoshua Bengio. Pointing the unknown words. *arXiv preprint arXiv:1603.08148*, 2016.

[17] Zichao Yang, Phil Blunsom, Chris Dyer, and Wang Ling. Reference-aware language models. *arXiv preprint arXiv:1611.01628*, 2016.

[18] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*, 2014.

[19] Yarin Gal and Zoubin Ghahramani. A theoretically grounded application of dropout in recurrent neural networks. In *Advances in neural information processing systems*, pages 1019–1027, 2016.

[20] Stanislau Semeniuta, Aliaksei Severyn, and Erhardt Barth. Recurrent dropout without memory loss. *arXiv preprint arXiv:1603.05118*, 2016.

[21] Hakan Inan, Khashayar Khosravi, and Richard Socher. Tying word vectors and word classifiers: A loss framework for language modeling. *arXiv preprint arXiv:1611.01462*, 2016.

[22] Daniel Jurafsky and James H. Martin. *Speech and Language Processing (2Nd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2009.

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

Authored by (in block letters):
For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.