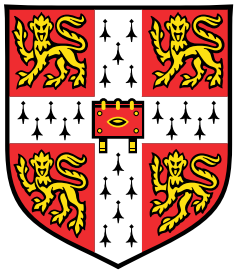
Improved Interpretability and Generalisation for Deep Learning



Mara Graziani

Department of Engineering
University of Cambridge

This dissertation is submitted for the degree of
Master of Philosophy

Pembroke College August 2017

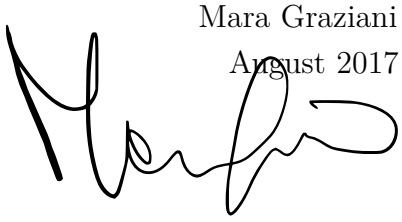
To who can think, who can wait, who can fast.
To my beloved family.

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements. This dissertation contains fewer than 15,000 words including appendices, bibliography, footnotes, tables and equations and has fewer than 150 figures.

Total word count: 12187 words
Signed: Mara Graziani
Date: 11th August 2017

Mara Graziani
August 2017



Acknowledgements

I would like to direct a big thanks to:
Professor Mark Gales, for the stimulating environment he allowed me to work in.
Andrey, Anton, Chungyang and Jeff, for the excellent support during the path.
The MPhil group for the great time together.
Feather Face, for always being there when fries are around.
Francesca, for being brave and following always my crazy ideas.
Bryan, for his support in the several nervous walks when nothing was working.
Alessandro, for distracting me in the evenings with his soul-touching music.
Finally and most importantly, my parents, for a long list of things.

This work would not have been of the same quality without all of you.

Grazie di cuore.

Abstract

Recurrent neural networks with gated units proved to be extremely powerful in sequence modeling. However, their generalisation and interpretability are still extremely challenging tasks. Improvements in generalisation and interpretability of deep feedforward networks for Automatic Speech Recognition have been achieved by using *Stimulated learning*. The same framework could be applied to Recurrent Language Models. In this work, we stimulate the internal activations of the gated units, which we reorganise first in a 2D grid and subsequently in a torus, to form a smooth surface. Moreover, we exploit interpretable word2vec stimuli to obtain a dynamic target-specific smoothing of the activations. The final performances result improved by the stimulation, showing that encouraging spatial smoothness yields better generalisation.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

In this thesis we expand the application of the *Stimulated Learning* framework [50] to Recurrent Neural Networks for Language Modeling (RNNLMs). The proposed stimulation organises the activations in a toroidal grid and introduces a spatial smoothing constraint on the activation surface. We then exploit *word2vec* target patterns to address interpretability. Experimentation has been carried out to assess the model performance compared to existing RNNLMs [8]. The models are then applied on an Automatic Speech Recognition task (ASR) to compute Word Error Rate performances. In most cases, the proposed model achieves comparable or better performances than the non-stimulated models.

1.1 Motivation

The *Stimulated training* of Deep Neural Networks (DNNs) was first proposed by Tan et al. in 2015 [44] and further investigated in [39] and [50], showing that introducing stimuli during training improves the robustness to unseen data, leading to better generalisation. The spatial smoothing stimulation of acoustic models yielded gains in WER for ASR [50, 51] and Keyword search [39], leaving the open question of whether the same stimuli could be beneficial to LM tasks.

In this work we analyse and expand the application of the framework to RNNLMs. We focus on the concept of spatial smoothing, proposing a novel reorganisation of the activation units in a toroidal surface. We then exploit *word2vec* stimulation patterns to address RNNLM interpretability. The key motivations to conduct this investigation are mainly two. First, it gives the opportunity to apply the stimulated learning framework on RNNs. Second, it allows to analyse the impact of switching the domain from acoustic modeling to language modeling, opening the discussion about stimulated learning to a broader range of sequential data inputs.

1.2 Organisation of the chapters

The remaining chapters are organised as follows:
In Chapter 2 we introduce the necessary background information about Deep Learning. In Chapter 3 we outline the main notions regarding Statistical Language Modeling and describe Neural Language Models and RNNLMs. In Chapter 4 we introduce the concept of Activation Based Regularisation and present Spatial Smoothing. Network Interpretability is addressed in Chapter 5. The experiments are described in Chapter 6, specifically the experimental results on spatial smoothing are reported in Section 6.3 and results on interpretability can be found in Section 6.5. Finally, in Chapter 7 the experimental results are discussed and directions for future work are suggested.

1.3 Claims of the thesis

The most important original contributions of this thesis are:

- Implementation of the Stimulated Framework for the TF-RNNLM [1]
- Analysis of the *spatial smoothing* regularisation and improvements in generalisation performances
- Analysis of *word2vec* and Part-of Speech stimulation patterns

[1] Source code location: https://github.com/mormontre/thesis

Chapter 2

Deep Learning

In this section we review the core concepts of deep learning architectures that will be further addressed in the dissertation, reporting the related literature to place the investigation within a broader context.

2.1 Feedforward Deep Neural Networks

Feedforward Neural Networks (NNs) attempt to approximate a function f, which maps the input x to some label category y: $y = f(\boldsymbol{x};\theta)$, where θ are the parameters of the approximation. NNs can have one or more internal layers (see Figure 2.1), which introduce a sequence of nonlinearities that are not accessible from the outside.

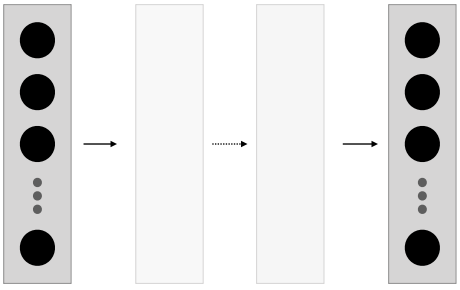


Fig. 2.1 *Feedforward deep neural network with multiple hidden layers*

For instance, the input of the $(l+1)$-th hidden layer, h^{l+1}, is a function of the ouptut of the l-th layer, $\boldsymbol{z}^l(x)$:

$$\boldsymbol{h}^{l+1}(\boldsymbol{x}) = \boldsymbol{\sigma}(z^l(\boldsymbol{x})) \tag{2.1}$$

$$\boldsymbol{z}^l(\boldsymbol{x}) = \boldsymbol{W}^l \boldsymbol{h}^l(\boldsymbol{x}) + \mathbf{b}^l$$

where $\boldsymbol{\sigma}(z)$ is the activation function and $\boldsymbol{W}$ and $\boldsymbol{b}$ are respectively the weight matrix and the bias of the transformation.
The output layer of the network for classification tasks is usually a softmax function, which is defined as:

$$\phi(\boldsymbol{x}) = \frac{exp(z_i(\boldsymbol{x}))}{\sum_{j=1}^{J} exp(z_j(\boldsymbol{x}))} \tag{2.2}$$

Activation functions The activation function determines the nature of the hidden units. The most common examples of activation functions are shown in Figure 2.2. The simplest is the step function, which is discontinuous in zero and is either equals to 1 or 0 elsewhere. Examples of more sofisticated activations are the sigmoid, the

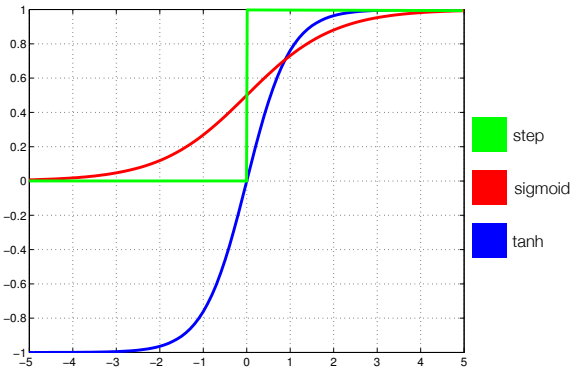


Fig. 2.2 *Common activation functions*

softmax and the hyperbolic tangent (*tanh*), which do not present discontinuities in zero. The sigmoid, or logistic regression function $\sigma(z)$ is defined as:

$$\sigma(\boldsymbol{z}) = \frac{1}{1+e^{-\boldsymbol{z}}} \tag{2.3}$$

The hyperbolic tangent function $tanh(z)$ is defined as:

$$tanh(\boldsymbol{z}) = \frac{e^{\boldsymbol{z}} - e^{-\boldsymbol{z}}}{e^{\boldsymbol{z}} + e^{-\boldsymbol{z}}} \tag{2.4}$$

The output of both the sigmoid and the hyperbolic tangent are continuous although they are defined over two different intervals, which are respectively [0,1] and [-1,1].

2.1.1 Network training

The training of neural networks consists of learning the set of parameter θ that minimises a loss function L. Two popular choices for the loss function are the Least Squares Error function and the Cross-Entropy criterion. Cross-Entropy is generally the training loss of networks with a softmax output over a number of possible classes k. For a set of training examples $\{\{\boldsymbol{x}_1, \boldsymbol{t}_1\}, \cdots, \{\boldsymbol{x}_n, \boldsymbol{t}_n\}\}$, Cross-Entropy is defined as:

$$L^{CE}(\theta) = -\sum_{i=1}^{n}\sum_{i=1}^{k} \boldsymbol{t}_i log\left(f(\boldsymbol{x}_n, \theta)\right) \tag{2.5}$$

where $\boldsymbol{t}_i$ is the target output for the input $\boldsymbol{x}_i$.

The computation of the gradients is efficiently performed using the **backpropagation** algorithm [21], although the nonlinearities in the network require the use of iterative gradient-based optimizers.

Gradient optimisation Gradient descent is the simplest and most used network training algorithm. The parameters are initialised with a starting guess and iteratively updated so that small steps are made in the direction of the greatest rate of decrease of the cost function L. A **momentum** term [38, 2] is generally added to the gradient descent formula, leading to the following expression:

$$\boldsymbol{\Delta}\theta^{\tau} = -\eta \boldsymbol{\nabla} L(\theta)|_{\theta^{\tau}} + \mu \boldsymbol{\Delta} L(\theta^{\tau-1}) \tag{2.6}$$

where the learning rate η specifies the convergence speed of the algorithm by controlling the impact of the gradient on the new estimates of the weights [4] and μ is the momentum parameter. The momentum adds inertia to the descent motion and reduces the number of oscillations towards the minimum as shown in Figure 2.3. The Nesterov Momentum

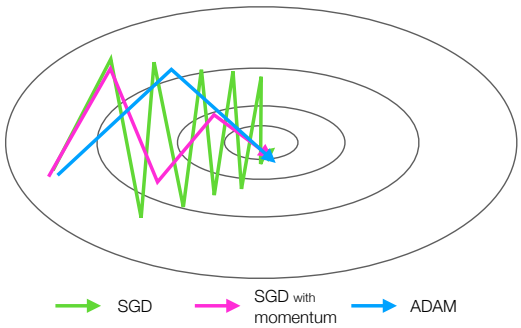


Fig. 2.3 *Comparison of Stochastic Gradient Descent algorithms*

further modifies the update rule by computing the gradient after the application of $\nabla L(\theta)|_{\theta^\tau}$ [43].

Adaptive learning rate optimisation The optimisation process is extremely sensible to the initial value of the learning rate parameter. Some optimisation algorithms such as AdaGrad, Adadelta[53], and RMS [46] adapt the learning rates individually for each model parameter. Adam, proposed by Kingma and Ba in [28], is based on adaptive estimates of lower-order moments and combines the benefits of both RMS and AdaGrad. Instead of adapting the parameter learning rates based as in RMSProp on the average first moment, e.g the mean, Adam includes in the adaptation the average of the second moments of the gradients, namely the variance. Specifically, the algorithm calculates an exponential moving average of the gradient and the squared gradient, and two parameters control the decay rates of the moving averages. The method is computationally efficient, has little memory requirements, is invariant to the diagonal rescaling of the gradients and is well suited for problems that are large in terms of data and parameters[28]. Moreover, Adam includes bias corrections to the estimates of the first order moments and the second order moments, taking account of the network initialisation at the origin.

Sensitivity to initialisation The initial values of the parameters are a relevant issue for a correct training process, since different initialisation may determine whether convergence is reached, at what speed, and the final value of the local optimum. To reduce the effects of different initialisations, two main observations shoud be taken into account. First, the symmetry between different units should be broken by adding Gaussian noise to the initial values. Second, the initial values of the weights should be set in such a way to prevent both the explosion and the vanishing of the gradients. For instance, they should be set neither too large neither too small. A sensible initialisation can be obtained with the Xavier initialisation[18] , which constrains the variance of the input gradient to be the same as the variance of the output gradient by imposing:

$$Var(\boldsymbol{W}_i) = \frac{1}{n_{out}} \tag{2.7}$$

where n_{out} is the number of neurons of the next layer.

2.2 Recurrent Neural Networks

RNNs are a family of neural networks for processing sequential data. The RNN structure is showed in Figure 2.4. The output is a function of the hidden layer $\boldsymbol{h}$, which is a recurrent layer. The recurrent layer is fed as input to the network together with $\boldsymbol{x}_i$:

$$\begin{aligned} \boldsymbol{h}_t &= f^h(\boldsymbol{W}_h^f \boldsymbol{x}_t + \boldsymbol{W}_h^r \boldsymbol{h}_{t-1} + \boldsymbol{b}_h) \\ \boldsymbol{y}_t &= f^f(\boldsymbol{W}_y \boldsymbol{h}_t + \boldsymbol{b}_y) \end{aligned} \tag{2.8}$$

where $\boldsymbol{W}_h, \boldsymbol{b}_h$, is the couple weight matrix, bias associated to the recurrent layer and $\boldsymbol{W}_y, \boldsymbol{b}_y$ is the one associated to the output. The main peculiarity of RNNs is that they share the same weights across the time steps through $\boldsymbol{h}_t$. As a result, RNNs scale to much longer sequences than DNNs.

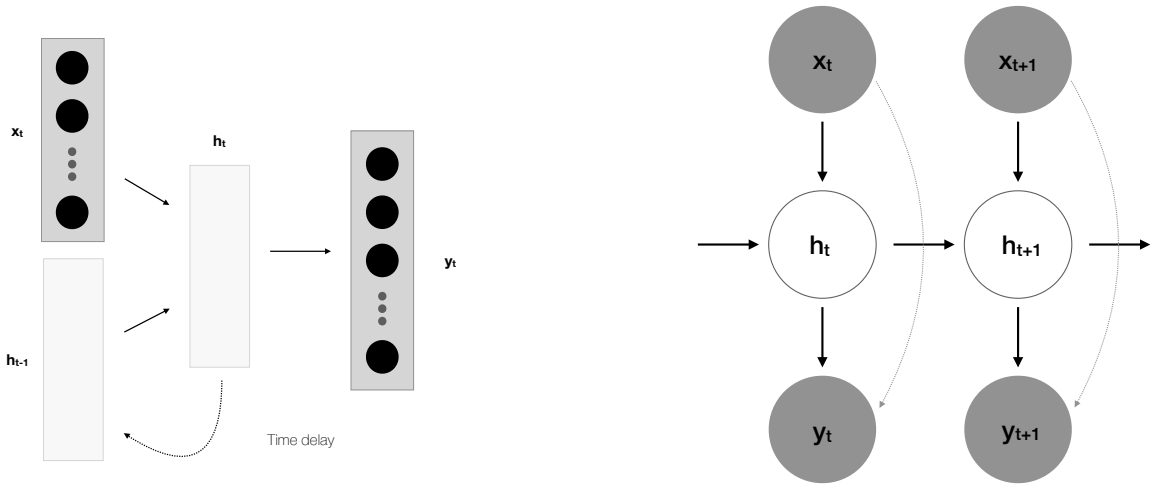


Fig. 2.4 *Recurrent Neural Network structure. Left:* compact representation. *Right:* RNN unfolded computational graph for two steps in time.

The computational graph of RNNs can be unrolled into directed acyclic graphs (DAG), which represent copies of the memory RNN cells connected forwards in time, as shown in Figure 2.4 (Right). The hidden state is passed throughout the sequence of computations as a temporary memory mechanism. For this reason, the hidden state is often referred to as *history vector*. Unrolling the RNN structure allows to efficiently compute the gradients by dividing their computation in a series of local gradients with backpropagation through time (BPTT)[49].

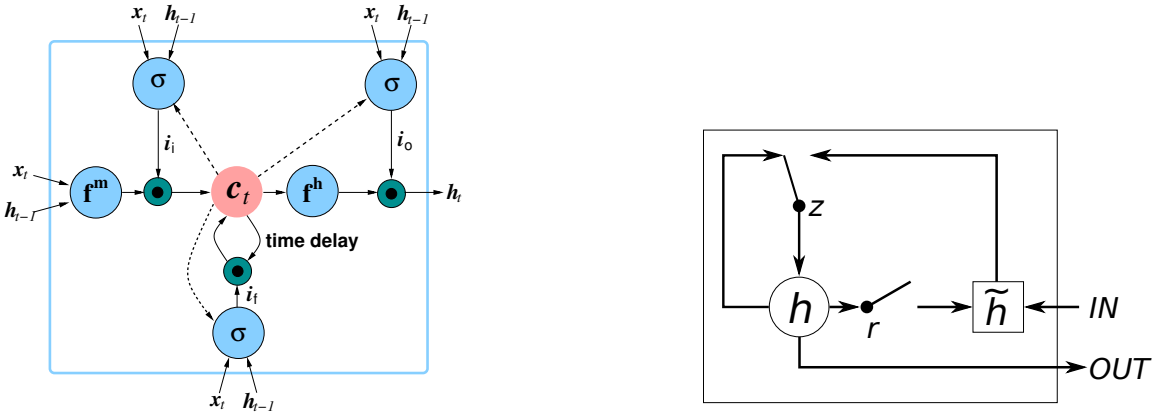


Fig. 2.5 *LSTM and GRU cells. Left:* Long Short Term Memory networks. Figure credits to [23]. *Right:* Gated Recurrent unit. Figure credits to [10]

Long-term dependencies Hochreiter in 1991 and Bengio et al. in 1993 showed that RNNs suffer of **vanishing** and **exploding gradients** when the dependencies are kept over many stages[23, 3]. The gradient of long-term interactions have exponentially smaller magnitude than the gradient of a short-term interactions and this causes the learning process to be either really slow or really unstable [20]. Gating solutions such as Long Short-term Memory (LSTMs) networks and Gated Recurrent Units (GRUs) (which are shown in Figure 2.5) create paths through time which avoid the derivatives to vanish or explode.

2.2.1 Long Short Term Memory networks

The LSTM model has been proposed by Hochreiter and Schmidhuber in 1997 and perfectioned by Gers et al in 2000 [23, 15, 14, 16]. In LSTM cells the input and the output are the same as in an ordinary recurrent neural network and a synergy of gating functions and memory controls the flow of information. The structure of the LSTM cell is shown in Figure 2.5 (on the left). The main component of the LSTM cell is the **state unit** $\boldsymbol{c}_t$, which represents the long-term memory of the cell. The information in the state unit is kept updated by three gates, namely the *input* gate, $\boldsymbol{i}_t$, the *output* gate, $\boldsymbol{o}_t$, and the *forget* gate, $\boldsymbol{f}_t$.

The forget gate resets from the memory the parts of information that are not relevant anymore to the learning process. The input and the output gate control respectively the amount of information that is currently relevant to the learning process and the information that should be kept as a candidate for the short term memory. Formally, the LSTM cell is defined as follows:

$$
\begin{aligned}
\boldsymbol{f}_t &= \boldsymbol{\sigma}_g(\boldsymbol{W}_f \boldsymbol{x}_t + \boldsymbol{U}_f \boldsymbol{h}_{t-1} + \boldsymbol{b}_f) \\
\boldsymbol{i}_t &= \boldsymbol{\sigma}_g(\boldsymbol{W}_i \boldsymbol{x}_t + \boldsymbol{U}_i \boldsymbol{h}_{t-1} + \boldsymbol{b}_i) \\
\boldsymbol{o}_t &= \boldsymbol{\sigma}_g(\boldsymbol{W}_o \boldsymbol{x}_t + \boldsymbol{U}_o \boldsymbol{h}_{t-1} + \boldsymbol{b}_o) \\
\boldsymbol{c}_t &= \boldsymbol{f}_t \circ \boldsymbol{c}_{t-1} + i_t \circ \boldsymbol{\sigma}_c(\boldsymbol{W}_c \boldsymbol{x}_t + \boldsymbol{U}_c h_{t-1} + \boldsymbol{b}_c) \\
\boldsymbol{h}_t &= \boldsymbol{o}_t \circ \boldsymbol{\sigma}_h(\boldsymbol{c}_t)
\end{aligned}
\tag{2.9}
$$

where $\circ$ denotes the Hadamard product (e.g. element wise product); $\boldsymbol{x}_t$ is the input vector; $\boldsymbol{h}_t$ is the output vector; $\boldsymbol{c}_t$ is the cell state vector; and $\boldsymbol{W}$, $\boldsymbol{U}$ and $\boldsymbol{b}$ are respectively the parameter matrices and bias vectors for each function update.

The activation functions $\boldsymbol{\sigma}_g, \boldsymbol{\sigma}_c$ and $\boldsymbol{\sigma}_h$ are usually hyperbolic tangents [42], although different alternatives have been proposed in [15] and [14].

2.2.2 Gated Recurrent Units

GRU cells share the same idea of LSTM cells, but have less gating functions. The structure of the GRU unit is shown in Figure 2.5. GRUs do not possess an internal memory and have only two gates, namely the *reset* gate and the *update* gate. The reset gate $\boldsymbol{r}_t$ determines how to combine the new input $\boldsymbol{x}_t$ with the previous memory $\boldsymbol{h}_{t-1}$, whereas the update gate $\boldsymbol{z}_t$ defines how much memory is important to keep.

Formally, the GRU unit is defines as:

$$
\begin{aligned}
\boldsymbol{z}_t &= \boldsymbol{\sigma}_g(\boldsymbol{W}_z \boldsymbol{x}_t + \boldsymbol{U}_z \boldsymbol{h}_{t-1} + \boldsymbol{b}_z) \\
\boldsymbol{r}_t &= \boldsymbol{\sigma}_g(\boldsymbol{W}_r \boldsymbol{x}_t + \boldsymbol{U}_r \boldsymbol{h}_{t-1} + \boldsymbol{b}_r) \\
\boldsymbol{h}_t &= \boldsymbol{z}_t \circ \boldsymbol{h}_{t-1} + (1 - \boldsymbol{z}_t) \circ \boldsymbol{\sigma}_h(\boldsymbol{W}_h \boldsymbol{x}_t + \boldsymbol{U}_h(\boldsymbol{r}_t \circ \boldsymbol{h}_{t-1}) + \boldsymbol{b}_h)
\end{aligned}
\tag{2.10}
$$

where $\boldsymbol{W}, \boldsymbol{U}$ and $\boldsymbol{b}$ are the relative parameter matrices and bias vector of each gate. The activation $\boldsymbol{\sigma}_g$ is a sigmoid function, whereas generally a *tanh* is used for $\boldsymbol{\sigma}_h$.

GRU and LSTMs are superior to the standard recurrent units although it is still not clear which of the two gating units is better [10]. On the one hand, the internal recurrency of LSTMs gives them more expressive power than the one of GRUs. On the other hand, the number of parameters in LSTMs is higher than in GRUs and this leads to a more complex model with slower training times.

2.3 Network Generalisation

Ensuring that the model will perform well not only on the training data but also on new inputs is a central problem in deep learning. *Generalisation* is defined as the ability to train with one data set and then successfully classify independent test sets, possibly reducing the test error. One way of achieving generalisation is through the introduction regularisation constraints in the objective function. In theory, the introduction of the correct regularisation to the objective function can ensure even very large and complex models to appropriately generalise. However, in practice really often the testing performances of conventionally regularised large models do not overcome markedly the training performances [55].

2.4 Regularisation

Regularisation is as any modification of the learning algorithm which is intended to reduce the generalisation error but not the training error. Some regularisation methods consist in the addiction of extra constraints on the models, with the purpose of restricting of placing soft boundaries to the space of the parameter values.

The regulariser $R(\theta)$ is added as a penalty to the cost function to introduce a trade-off between fitting the training and satisfying a further condition. The original objective function L is modified as follows:

$$\tilde{L}(\theta) = L(\theta) + \gamma R(\theta) \tag{2.11}$$

where $\gamma \in [0, \infty)$ is a hyperparameter that weights the regularisation contribution.

There is no an overall best form of regularisation, since each task usually requires different additional constraints. Very often, parameter norm penalties, e.g. L1, L2, are used to limit the capacity of the models while achieving specific objectives such as weight decay or weight sparsity. In the following section we briefly analyse the effect conventional regularisation methods.

2.4.1 L2: weight decay

One of the simplest and most common kinds of parameter norm penalty is L2, which is also known as weight decay. The L2 regularisation is a norm regularisation term which keeps the values of the weights as small as possible during the optimisation. For

instance, under the no bias parameter assumption ($\theta = w$), the L2 regularisation can be written as:

$$R^{L2}(\theta) = \boldsymbol{w}^T \boldsymbol{w}$$

The training criterion is modified in:

$$L(\theta) = \frac{\gamma}{2} \boldsymbol{w}^T \boldsymbol{w} + L(\theta) \tag{2.12}$$

with corresponding parameter gradient:

$$\boldsymbol{\Delta}_w L(\theta) = \gamma \boldsymbol{w} + \boldsymbol{\Delta}_w L(\theta) \tag{2.13}$$

To take a single gradient step to update the weights, the update rule is as follows:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \epsilon(\gamma \boldsymbol{w} + \boldsymbol{\Delta}_w L(\theta)) \tag{2.14}$$

Which leads to the following rule:

$$\boldsymbol{w} \leftarrow (1 - \epsilon\gamma)\boldsymbol{w} - \epsilon \boldsymbol{\Delta}_w L(\theta) \tag{2.15}$$

Eq. 2.15 shows the shrinking effect on the weight vector by the constant factor $(1 - \epsilon\gamma)$ at each iteration [20]. L2 causes the learning algorithm to 'perceive' the input as having higher variance, which makes it shrink the weights on the features whose convariance with the output target is low compared to this added variance[20].

2.4.2 L1: weight sparsity

Another traditional regularisation is L1, which can instead be interpreted as promoting sparsity in the weight matrix. L1 regularisation is defined as the sum of the absolute values of the individual parameters:

$$\omega(\theta) = ||w||_1 = \sum_i |w_i| \tag{2.16}$$

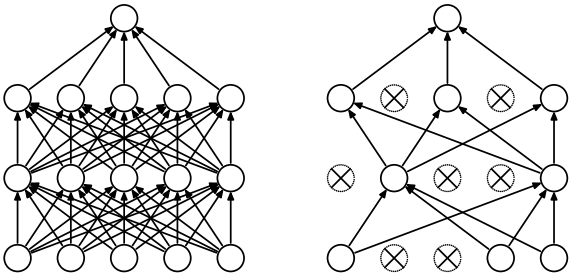
The optimisation function becomes:

$$\tilde{L}(\theta) = \gamma ||\boldsymbol{w}||_1 + L(\theta) \tag{2.17}$$

The gradient of $||\boldsymbol{w}||$ generates a $sign(\boldsymbol{w})$ term in the update rule, which results in sparse weight solutions.

2.4.3 Dropout: ensemble of infinite subnetworks

Dropout has a powerful regularisation effect, although it is not applied directly as a regularisation term in the training objective. Conceptually, dropout provides an inexpensive approximation to training and evaluating an ensemble of exponentially many neural networks. The trained ensemble consists of all subnetworks that can be formed by removing non output units from an underlying base network.



(a) Standard Neural Net (b) After applying dropout.

Fig. 2.6 *Dropout Neural Network.* Figure credits to: [41] *Left*: A standard NN with 2 hidden layers. *Right*: Example of the 2 hidden layer network after the application of dropout. Crossed units have been dropped.

In Figure 2.6 we report the illustration of the working principle of dropout proposed by Srivastava et al. in [41]. Each node is assigned a probability to be used or discarded. The crossed nodes have been dropped from training, hence reducing the computational power of the network.

Chapter 3

Language Modeling

In this chapter we briefly report the background notions on statistical language modeling and illustrate the main characteristics of RNNLMs.

3.1 Statistical Language Modeling

Statistical Language Modeling is a sequential data modeling problem, whose goal is to capture the information about the organisation of the words in a sentence. Given a sentence $\boldsymbol{\omega}_{1:L} = \{\omega_1, \omega_2, .., \omega_L\}$, its structure can be modeled as a prediction problem. For instance, $\boldsymbol{\omega}_{1:L}$ can be seen as a sequence of words in the vocabulary, and we can estimate the probability distribution of the next word. The probability distribution reflects how likely is any given word to occur next in the sequence.
Formally, we want to compute the joint probability $p(\boldsymbol{\omega}_{1:L})$:

$$p(\boldsymbol{\omega}_{1:L}) = p(\omega_1, \omega_2, \ldots, \omega_L) \tag{3.1}$$

Which can be decomposed into a product of conditional probabilities of the next word, given the previous history:

$$p(\boldsymbol{\omega}_{1:L}) = \prod_{i}^{L+1} p(\omega_i | \omega_1, \omega_2, \ldots, \omega_{i-1}) \tag{3.2}$$

where sentence start and sentence end markers are often added to the word sequences. The Markov assumption is used to condition the probability only on the previous N words, since handling all possible word sequences is impractical in both terms of computational complexity and storage. The resulting model is called N-gram model.

3.1.1 N-gram models

N-gram language models are the standard approximation to making the calculation of the word sequence probability practicable. A general N-gram model allows to compute the probability of the joint sequence as a function of the conditional probabilities of the $n-th$ token given the preceding $N-1$ as follows:

$$p(\omega_1,\omega_2,\ldots,\omega_L) = \prod_{i}^{L+1} p(\omega_i|\omega_i-1,\omega_i-2,\ldots,\omega_{i-N+1}) \tag{3.3}$$

N-grams training Training N-grams is straightforward with maximum likelihood (ML) estimation, since the ML estimate can be computed by counting the number of times a certain N-gram appears in the training corpus.

Limitations of N-gram models N-gram models allow an efficient computation of a baseline model. However, they have a number of drawbacks. First, they have poor generalization capabilities. For instance, in case of unseen or rare words, the ML probability estimates tend to collapse to zero. Smoothing and back-off methods are generally used to eliminate the consequences of assigning zero probability to uncommon or unseen words in the training set [7]. Second, classical N-gram models are vulnerable to the curse of dimensionality. The number of possible n-grams scales exponentially with the size of the vocabulary, which is generally very large. Finally, N-grams are discrete and sparse. Their computation does not take into account any similarity between words neither at the semantic or the grammatical level, ignoring a fundamental part of language structure.

Neural Language models overcome most of these issues and therefore are generally preferred to N-grams.

3.1.2 Neural Language Models

The DNNLM proposed by Bengio is shown in Figure 3.2. The input words are defined over finite length vocabulary V and are represented using a 1-of-V coding. For instance, each word is associated with a vector of length V where all elements are zero, except the value corresponding to the index of the given word. The first part of the network finds a linear projection matrix P that maps the one-hot encoded words to a real vector of a smaller dimensionality m, called *distributed feature vector*. The projection matrix P is sharedd among words at different positions in history and models words similarities as clusters[3]. In the second part of the network a hidden layer with a non-linear

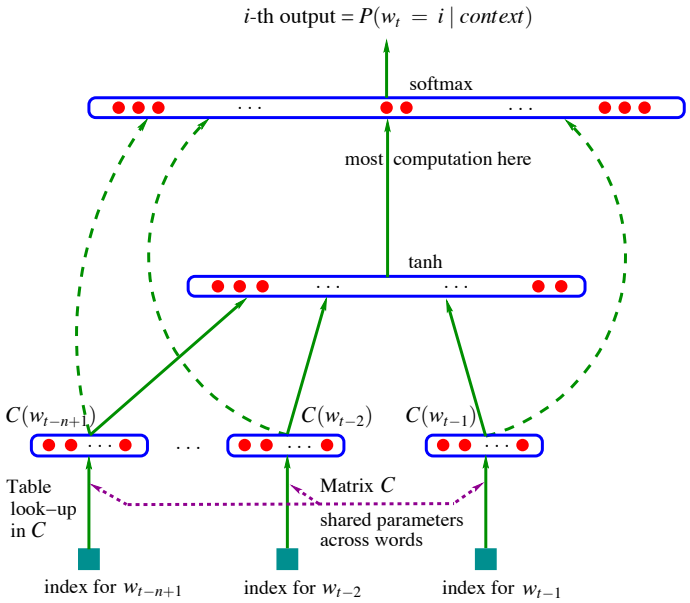


Fig. 3.1 *Bengio's Neural Language Model.* Figure credits [3].

activation function (generally a *tanh* or a *sigmoid*) computes a transformation of the input (generally of 100-300 dimensions), and finally a softmax over the size of the full vocabulary is used to produce the output predictions. The output layer represents the probability distribution $P(\omega_t|\boldsymbol{h}_{t-1})$ [35].

Distributed representation of words The main advantage of Neural Language Models over the standard $N-$gram model is the projection layer given by the transformation P. The projection operation maps the sparse one-hot vectors to a distributed representation.

The concept of distributed representation was first presented by Hinton in 1984 [22]. The distributed space allows to represent one concept with many units, which may contribute simultaneously to the representation of more than just one concept. This formulation is extremely powerful and compact, since it allows to use n features with k values to describe k^n different concepts.

Hinton's original practical example is useful to show how the distributivity of the space enhances generalisation to similar concepts. In a distributed representation framework chimpanzees and gorillas are likely to be close in the representation since they share similar features. Given the prior knowledge that chimpanzees like onions, then in the distributed space the expectation that also gorillas like onions arises.

The point of strenght of the use of such representation for words is that it allows to *leave the sparse one-hot representation* and model language in a *dense and flexible space*. In such space words with similar meanings or referring to common contexts are mapped very close to each other. As a result, one single concept can be differentiated into several slightly different concepts by subsequent modifications in the weight parameters [22]. Notwithstanding, the mapping between the concept and its representation is not a bijection, hence is not directly invertible. This makes the interpretation of the representations extremely challenging (see Section 5.1).

3.2 Recurrent Neural Network Language Models

The structure of RNNLMs is shown in Figure 3.2. The model maintains the same 1-hot coding and the layer projection [3]. The one-of-K representation is generally done over the entire vocabulary or over the subset of the most frequent K words, called shortlist, to save computational time. The words projections are computed as in Bengio's neural model and then fed together with the history vector as input to the network. The hidden layer compresses the information from the two inputs and computes a new representation using a sigmoid activation. The output layer is given by a further softmax, which produces the normalised RNNLM probabilities for the next word given the history [8]. The probability of the word sequence is given by:

$$P(\boldsymbol{\omega}_{1:L}) = \prod_{i=1}^{L+1} P(\omega_i|\boldsymbol{\omega}_{1:i-1}) \approx \prod_{i=1}^{L+1} P(\omega_i|\omega_{i-1}, \boldsymbol{h}_{i-2}) \approx \prod_{i=1}^{L+1} P(\omega_i|\boldsymbol{h}i-1) \tag{3.4}$$

Training Cross Entropy criteria The standard cross entropy training criteria in Eq. 2.5 for the word sequence $\boldsymbol{\omega}_{1:L}$ is computed on the word sequence as follows:

$$L^{CE}(\theta) = -\frac{1}{L}\sum_{i=1}^{L} log(P_{RNN}(\omega_i|\boldsymbol{h}_i)) \tag{3.5}$$

where the RNN output probability is given by the softmax activation function.

Advantages of a recurrent LM The main difference between the feedforward NLMs and RNNLMs is in representation of the history. For feedforward NNLMs, the history is just a representation of the previous words, whereas in RNNLMs the representation of history is learned from the data during training. This allows a more

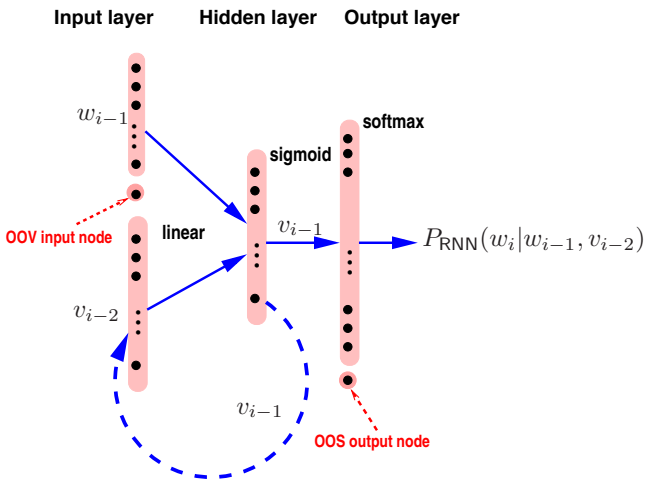


Fig. 3.2 *CUED RNNLM*. Figure credits to [9].

efficient encoding of words that could have occurred at variable position in the history, since the model can simply remember the older parameters learned for the previous occurrence of that word. Conversely, feedforward NNs would have to recompute the values of the parameters each time.

3.3 Evaluation Metrics and application to ASR

The evaluation of LMs can be done either in an intrinsic or in an extrinsic fashion. The intrinsic evaluation involves the computation of the model entropy and the relative perplexity. The extrinsic evaluation requires the application of the model to a specific task, which can be for example rescoring the word probabilities of the output of an ASR system and compute the Word Error Rates (WERs) with respect to a reference.

3.3.1 Perplexity

Perplexity is defined as:

$$PPL = P(\boldsymbol{\omega}_{1:L})^{-\frac{1}{L+1}} = \left(\prod_{i=1}^{L+1} P(\omega_i|\omega_1, .., \omega_{i-1}) \right)^{-\frac{1}{L+1}} \tag{3.6}$$

where L is the total number of words in the sentence $\boldsymbol{\omega}_{1:L}$.

Perplexity can informally be thought as a measure of how likely a language model is to predict the test data, since it can be seen as the exponential of the average

per-word cross entropy of the test data. There is a close relationship between entropy and perplexity, namely:

$$H = log_2 PPL$$

The model which yields the lowest perplexity is in some sense the closest to the true model which generated the data. Therefore, we can consider entropy and perplexity as very useful measures, which are strongly positively correlated with the final ASR system performance [35].

3.3.2 Word Error Rates

In the extrinsic evaluation the model is applied to the practical task of rescoring the acoustic output of an ASR system.

N-best lists The output of an acoustic model is usually a large number of candidate sentences, called *N-best lists*. Each sentence in the list has assigned a probability score, which is then used by the LM to reorder the different possibilities according to how likely is for each of them to belong to the language. For example, the sentence *'I saw a van'* sounds really similar to *'eyes awe of an'*, although the first should be preferred to the second because it conveys a meaningful message, whereas the other has no sense at all. The role of the LM is to rescore the probability of the two sentences and increase the likelihood of the first option with respect to the second.

Word Error Rates The WER is indicative of the mismatches between the best decoded utterance option W_{dec} and the reference utterance transcription W. After rescoring, the first best option for each sentence is taken as the final output of the system and compared to an original reference transcription. Words and sequences are matched using dynamic programming, yielding the optimal string match which has the lowest possible score [52]. Once the optimal alignment is found, the number of substitution errors (S), deletion errors (D) and insertion errors (I) is calculated, and the WER is computed as:

$$WER = \frac{S + D + I}{N} \tag{3.7}$$

Language Model interpolation Language model interpolation is often used to merge the characteristics of two or more language models, which may express different aspects of language. The interpolation between two models a and n is computed follows:

$$P(\omega) \approx \prod_{i=1}^{L+1} (\lambda_a P_a(\omega_i|\omega_1,..,\omega_{i-1}) + \lambda_n P_n(\omega_i|\omega_1,..,\omega_{i-1}) \tag{3.8}$$

where the interpolation weights λ_a and λ_n reflect how appropriate is the language model for the target domain and are subject to $\lambda_a + \lambda_n = 1$. They can either be tuned manually or estimated with Expectation Maximisation to minimise the perplexity on a small amount of data from the target domain.

Generally, perplexity on held-out test data is preferred during the development stage, since the extrinsic evaluation has a series of drawbacks. First of all, it can be computationally expensive to test different language models on ASR systems. Moreover, it does not give and independent evaluation of the model, since performance is related to the system the model is applied to. Therefore usually perplexity on held-out test data is preferred during the development stage. The extrinsic evaluation has different drawbacks. WER does not give and independent evaluation of the model, since performance is related to the system the model is applied to. Therefore, comparisons between different setups are not particularly informative and only comparisons for the same system and data should be used. However, WER comparisons for the same system and data are excellent measures of performance, better than perplexity.

Chapter 4

Activation based regularisation

The main concept of stimulated learning lays in the introduction of stimuli during the network training process through the use of activation based regularisation terms.

In this chapter we review the basic principles of the stimulated framework and propose the Toroidal Spatial Smoothing regularisation term which encourages the formation of a smooth activation surface on a toroidal reorganisation of the network activations.

4.1 The Stimulated Learning framework

The Stimulated learning framework proposes a general procedure to obtain activation based regularisation terms that can be used to insert additional constraints on the values of the activations. Such constraints should be used to promote behaviours of the activations that are likely to improve both the interpretability and the generalisation of the network.

The main steps in stimulated training are four, namely the the organisation of the activations in an *activation surface*, the transformation of the activation with a specific transformation function, the definition of target patterns and the final definition of the regularisation term as a distance function between the current values of the activations and the desired targets.

4.1.1 Node organisation

The distributed nature of the embeddings allows the reorganisation the hidden activations according to any possible ordering. The work in [44] proposed the organisation in d-dimensional grids through the following operation:

$$G(\boldsymbol{x}_t, \theta) = \left\langle \boldsymbol{H}_t^{(1)}, \boldsymbol{H}_t^{(2)}, .., \boldsymbol{H}_t^{(L-1)} \right\rangle \tag{4.1}$$

where $\boldsymbol{H}_t$ is the activation vector grid at step t. In a 2-dimensional space $\boldsymbol{H}_t$ can be formalised as a matrix, whereas for $d > 2$ it is a higher dimensional tensor. For instance, the N activations $\boldsymbol{h}_i$ are reorganised in a$\sqrt{N} \times \sqrt{N}$ open grid **flat surface**:

$$\boldsymbol{h}_i = \begin{bmatrix} \boldsymbol{h}_{i,1} \\ \boldsymbol{h}_{i,2} \\ \vdots \\ \boldsymbol{h}_{i,N} \end{bmatrix} \rightarrow \begin{bmatrix} \boldsymbol{h}_{i,1} & \boldsymbol{h}_{i,2} & \dots & \boldsymbol{h}_{i,\sqrt{N}} \\ \boldsymbol{h}_{i,\sqrt{N}+1} & \boldsymbol{h}_{i,\sqrt{N}+2} & \dots & \boldsymbol{h}_{i,2\sqrt{N}} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{h}_{i,N-\sqrt{N}} & \boldsymbol{h}_{i,N-\sqrt{N}+2} & \dots & \boldsymbol{h}_{i,N} \end{bmatrix} \tag{4.2}$$

In the surface each activation is projected on a Cartesian coordinate system, thus each neuron can be located as a point in the network-grid space.

4.1.2 Activation Transformation

Three different types of activation transformation have been proposed in the previous work on stimulated learning [44, 51, 50], namely the activation normalisation, the filtering of the activations, and the transformation into a probability mass function.

4.1.3 Normalisation

The normalised activation is defined as:

$$\tilde{h}_t i^{(l)} = h_t i^{(l)} \beta_i^{(l)} \tag{4.3}$$

where $\beta_i^{(l)}$ is used to reflect the impact that the activation function has on the following layer $l+1$:

$$\beta_i^{(l)} = \sqrt{\sum_k w_{ik}^{(l+1)^2}} \tag{4.4}$$

The normalisation can be used to describe the activation behaviour and identify useless activations. For instance, $\beta_i^{(l)}$ is small when the weights of the next layer for the activation node i are small or for the neurons that only stay close to the extreme ends of the range of the activation function and that do not vary much during training. Since these neurons do not strongly contribute to the learning process, they can be removed by the representation.

4.1.4 Probability mass function

The space of the grid can be interpreted as a discrete probability space. Therefore, the activations can be transformed into a probability mass function, which will contain information about the activation behaviour. The probability mass function can be defined as follows:

$$\tilde{h}_{ti}^{(l)} = \frac{h_{ti}^{(l)}}{\sum_j h_{tj}^{(l)}} \tag{4.5}$$

In this representation the activations can be considered as a whole, instead of independent items.

4.1.5 Filtering

The d-dimensional activation surface can be interpreted as an *'activation image'* made of *'pixel neurons'* [50]. Digital Image Processing techniques such as filtering can be applied to the activation image. In 2D and 3D an immediate visualisation of the results can be obtained. For example, the application of linear smoothing filters replaces each activation pixel by a combination of its neighboring pixels, thus smearing out the activation image [39, 51].

Formally, linear filters are defined as a discrete two dimensional real valued function $K : \mathbb{Z}x\mathbb{Z} \rightarrow \mathbb{R}$, which can be represented as a filter matrix, $\boldsymbol{K}(i,j)$. The size of the matrix is equal to the size of the filter region, and every element $\boldsymbol{K}(i,j)$ specifies the weight of the corresponding pixel in the summation. The application of the filter to the activation image is done through a convolution process:

$$\tilde{\boldsymbol{H}}_t^{(l)} = \boldsymbol{H}_t^{(l)} \star \boldsymbol{K} \tag{4.6}$$

For instance, $\boldsymbol{K}$ is moved over the activation image $\boldsymbol{H}$ so that the origin $\boldsymbol{K}(0,0)$ corresponds to the current image position (u,v). The filter coefficients $\boldsymbol{K}(i,j)$ are then multiplied with the corresponding activation image element $\boldsymbol{H}(u+i, v+j)$ and the results are added together. Finally, the resulting sum is stored at the current position in the new image.

4.1.6 Target pattern

The activation target patterns $\boldsymbol{G}_t^{(l)}$ define the desired behaviour in the network that the stimulation should enhance. Different concepts can be modelled by different target patterns, which can be mainly distinguished in *time-variant* and *time-invariant*.

Time invariant patterns Time invariant patterns introduce a general concept which should be statically satisfied during the training of the network:

$$\boldsymbol{G}_t^{(l)} = \boldsymbol{G}^{(l)} \tag{4.7}$$

Time-variant targets Time variant patterns introduce dynamic constraints during training. The phoneme targets in [50] are an example of time variant patterns. Target areas are defined on the activation grid for a series of inputs. According to the current input the target is changed accordingly to promote to local activation of the target area. As a result, the stimulated training enables to learn representations that yield high activations for any given input only in the vicinity of the target zone for the specific input [39].

4.1.7 Regularisation

Regularisation aims at minimising the difference $D(\tilde{\boldsymbol{H}}_t^{(l)}, \boldsymbol{G}_t^{(l)})$ between the transformed activation $\tilde{\boldsymbol{H}}_t^{(l)}$ and the target pattern $\boldsymbol{G}_t^{(l)}$ over the training data:

$$R(\theta) = \frac{1}{T} \sum_t \sum_l D(\tilde{\boldsymbol{H}}_t^{(l)}, \boldsymbol{G}_t^{(l)}) \tag{4.8}$$

The difference between the transformed activations and the targets can be computed in three different ways, namely by using the Mean Squared Error, the KL-divergence or the Cosine Similarity.

Mean Squared Error The Mean Squared Error (MSE) method directly defines the regularisation term:

$$D(\tilde{\boldsymbol{H}}_t^{(l)}, \boldsymbol{G}_t^{(l)}) = ||\tilde{\boldsymbol{H}}_t^{(l)} - \boldsymbol{G}_t^{(l)}||_F^2 \tag{4.9}$$

where $||\boldsymbol{H}||_F$ is the Frobenious norm of the matrix H.

KL divergence The KL-divergence method uses the activations transformed in a probability mass function and compares them with a target distribution:

$$D(\tilde{\boldsymbol{H}}_t^{(l)}, \boldsymbol{G}_t^{(l)}) = \sum_i \tilde{g}_{ti} log(\frac{\tilde{g}_{ti}}{\tilde{h}_{ti}}) \tag{4.10}$$

The mismatch between the two distributions is minimised during the training process. As a result, the activation distribution is encouraged to form the target pattern [39].

Cosine similarity The use of cosine similarities extends the different concept to arbitrary activation types, without the necessity of a probability mass function. The cosine similarity is formally defined as:

$$\cos(\theta) = \frac{\tilde{\mathbf{H}}_\mathbf{t} \cdot \mathbf{G}_\mathbf{t}}{\|\tilde{\mathbf{H}}_\mathbf{t}\|_2 \|\mathbf{G}_\mathbf{t}\|_2} = \frac{\sum_{i=1}^{n} \tilde{h}_{ti}^{(l)} g_{ti}^{(l)}}{\sqrt{\sum_{i=1}^{n} \tilde{h}_{ti}^{(l)2}} \sqrt{\sum_{i=1}^{n} g_{ti}^{(l)2}}} \tag{4.11}$$

The cosine similarity encourages the similarity between two vectors in the inner product space, not requiring the transformation of the activation into a probability mass function. Instead of comparing the two probability distributions it measures the angle between the transformed activations vector and the target pattern.

4.2 Introducing Spatial Smoothing for LMs

The idea behind encouraging spacial smoothness is inherent in the definition of the *smoothness*, or local constancy of a function f, which assumes only small changes of the function values which lay within a small region. Therefore, if $u \approx v$, then the target function f should have the property $f(u) \approx f(v)$. Formally, we would like that:

$$f(x + \epsilon d) \approx f(x) \tag{4.12}$$

for unit d and small ϵ [20]. Eq. 4.12 assumes the similar behaviour of nearby input representations. This assumption perfectly matches the characteristics of the distributed representation of words in the projection layer of neural LMs. Therefore, enforcing spatial smoothing would benefit the spatial reorganisation of the hidden neurons and exploit the distributed nature of the network activations to improve the regularisation capabilities of the model. For example, we can consider the case of 'CAT' and 'DOG', which belong to the same semantic class of domestic animals. Assuming that their

representations are close in the distributed space, the activations for those inputs should be similar. The smoothing regularisation should therefore drive the difference between the activations relative to each input very close to zero. This can be achieved by the convolution of the activation surface with a high-pass filter.

4.2.1 High-pass filtering the activation image

Smoothing filters can be used to interpolate the values of nearby activation pixels. For instance, the simplest high-pass filter is the 3x3 box kernel (visualised in Figure 4.1), which is defined as:

$$K = \frac{1}{8}\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \tag{4.13}$$

The high-pass filtered image is obtained throught the convolution operation between the activation image and the filter, as in Eq. 4.6.

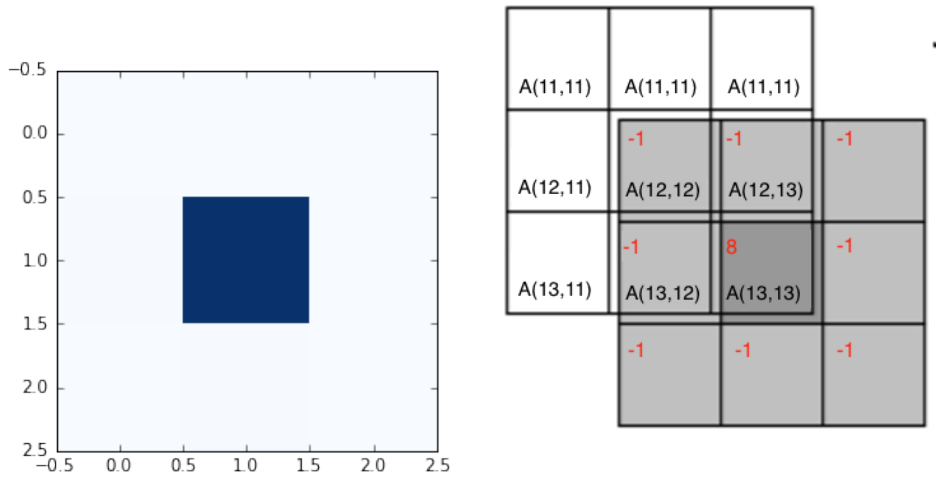


Fig. 4.1 *Filtering details. Right:* visualisation of a 3x3 box filter kernel. *Left:* convolution operation of the box filter with the activation grid.

Regularisation term The final regularisation term is given by encouraging the high-pass filtered activations to be small, namely by imposing the Frobenious norm of the activations to be close to zero. Formally:

$$R^{Smooth}(\theta) = \frac{\gamma}{N_w}||\boldsymbol{H}_t^l \star \boldsymbol{K}||_F^2 = \frac{\gamma}{N_w}||\tilde{\boldsymbol{H}}_t^l||_F^2 \tag{4.14}$$

where γ is the regularisation penalty, N_w is the total number of words in the vocabulary, and $\tilde{\boldsymbol{H}}_t^l$ are the filtered activations of the l-th layer at iteration t .

The training criterion becomes:

$$L(\theta) = L^{CE}(\theta) + \gamma R^{Smooth}(\theta) \tag{4.15}$$

where γ is a regularisation penalty, that controls the contribution of the activation regularisation term.

4.2.2 Filtering Edge-Effects

The borders of the activation grid represent a limit case that is a well-known issue in Image processing. Theoretically,the convolution can only be applied at locations (u, v) where the filter matrix $\boldsymbol{K}$ is fully contained in the image. Several solutions have been introduced, such as zero-padding or mirroring. Zero padding adds an additional border of zeros around the image, which allows to compute the convolutions.

As a side-effect, the 'zero' added context is smoothed with activations of the network, which results in driving the activations to very low values. For instance, Figure 4.2 we present the result of high-pass filtering a flat activation image, e.g. all the activations have been set to one, which represents an already perfectly smooth surface. The expected result of the filtering should be flat to zero, since there is no variation to smooth. The different colours at the borders show instead that edge-effects modify the value of the grid at the borders.

4.2.3 Changing the topology: from open to closed surfaces

An alternative solution is given by the introduction of curvature in the reorganisation of the hidden units to pass from the open, flat grid surface to a closed surface representation.

Closed surfaces Closed surfaces are *compact* and *unbounded*. Compactness is a property that generalizes the notion of a subset of Euclidean space being closed, namely containing all its limit points and that has all its points lying within some fixed distance between each other. Examples of closed surfaces are the sphere, the torus and the connection of a number of planes (eg. cube). The deformation of a flat grid into a torus through the introduction of contiguity is generally used in Digital Electronics, especially for logic simplification with the Karnaugh mapping [26].

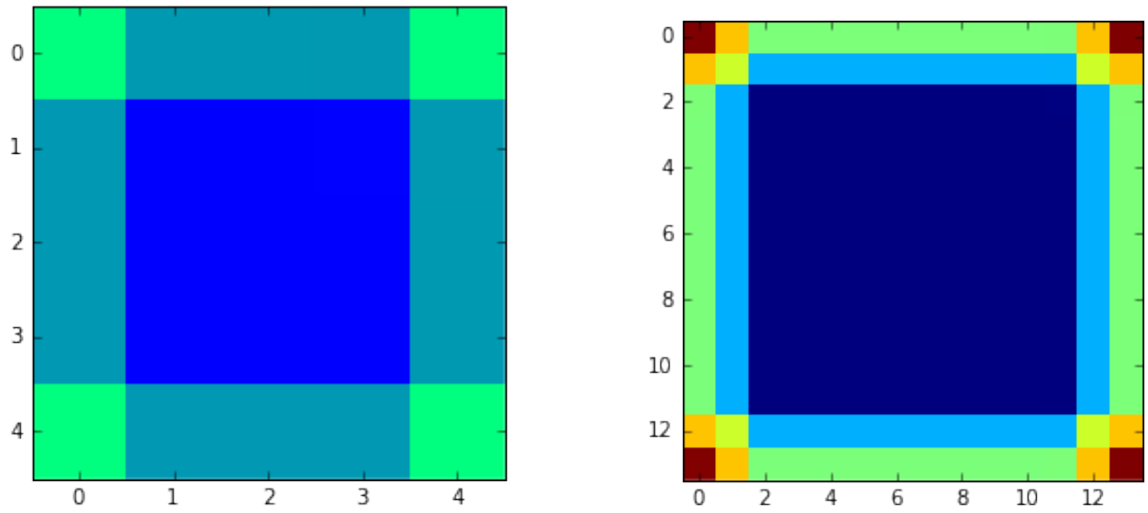


Fig. 4.2 *Edge effects at the grid borders. Left:* Edge effects after the convolution with a 3x3 Kernel of a totally smooth surface.The blue color corresponds to a perfectly smooth surface, with all zeros. The lighter colors at the borders show that the surface is perturbed by the introduction of zeros in the convolution. *Right:* 5x5 box filter kernel edge effects. As the size of the filter increases the side-effects at the edges are amplified.

Toroidal grids Topologically, a torus is a closed surface defined as the product of two circles: S1 × S1. In three dimensions, a flat grid can be deformed into a torus by joining the borders of the grid as shown in Figure 4.3. Toroidal grids can be obtained by applying the same principle used in the karnaugh maps. Adjacency is added as shown in Figure 4.3, where the nodes with the same colors are joined to form a closed surface.

Formally, the torus structure can be obtained modifying the node activations organisation in Eq. 4.2 as follows:

$$\boldsymbol{h}_i = \begin{bmatrix} \boldsymbol{h}_{i,1} \\ \boldsymbol{h}_{i,2} \\ \vdots \\ \boldsymbol{h}_{i,N} \end{bmatrix} \rightarrow \left[\begin{array}{ccccc|c} \boldsymbol{h}_{i,\sqrt{N}} & \boldsymbol{h}_{i,N-\sqrt{N}} & \boldsymbol{h}_{i,N-\sqrt{N}+2} & \dots & \boldsymbol{h}_{i,N} & \boldsymbol{h}_{i,N-\sqrt{N}} \\ \hline \boldsymbol{h}_{i,\sqrt{N}} & \boldsymbol{h}_{i,1} & \boldsymbol{h}_{i,2} & \dots & \boldsymbol{h}_{i,\sqrt{N}} & \boldsymbol{h}_{i,1} \\ \boldsymbol{h}_{i,2\sqrt{N}} & \boldsymbol{h}_{i,\sqrt{N}+1} & \boldsymbol{h}_{i,\sqrt{N}+2} & \dots & \boldsymbol{h}_{i,2\sqrt{N}} & \boldsymbol{h}_{i,\sqrt{N}+1} \\ \dots & \vdots & \vdots & \ddots & \vdots & \dots \\ \boldsymbol{h}_{i,N} & \boldsymbol{h}_{i,N-\sqrt{N}} & \boldsymbol{h}_{i,N-\sqrt{N}+2} & \dots & \boldsymbol{h}_{i,N} & \boldsymbol{h}_{i,N-\sqrt{N}} \\ \hline \boldsymbol{h}_{i,\sqrt{N}} & \boldsymbol{h}_{i,1} & \boldsymbol{h}_{i,2} & \dots & \boldsymbol{h}_{i,\sqrt{N}} & \boldsymbol{h}_{i,1} \end{array}\right] \tag{4.16}$$

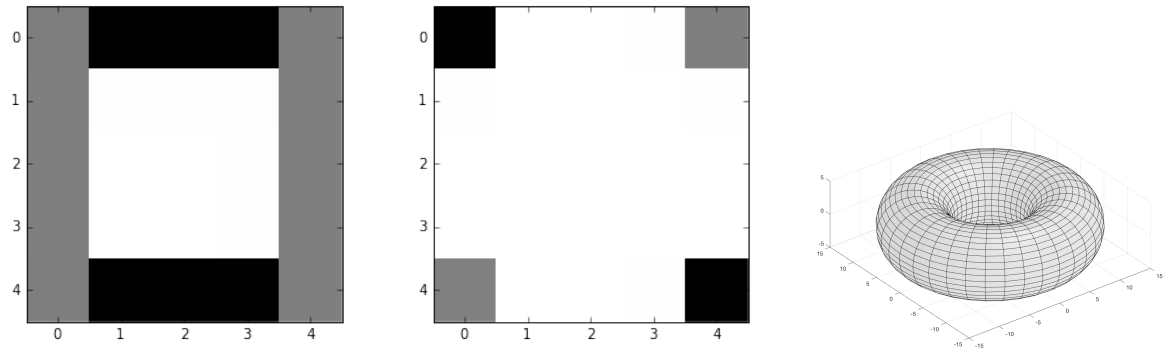


Fig. 4.3 *Deformation into torus. Left:* Adjacency is added on the horizontal and vertical evidenced stripes. The same colours are merged. *Middle:* the grid corners with the same colors are put in adjacent places. *Right:* resulting torus.

An expanded activation grid is created where one line is added on top, one line at the bottom and two columns are added to each side. The added line at the top replicates the values of the last time at the bottom, and the added line at the bottom replicates the values in the first line of the original activation grid. The same pattern is followed for the lateral columns. The values in the four angles are swapped so that the top left corner contains the value of the grid in the bottom right corner.

4.2.4 Relationship between Spatial Smoothness and L2

Encouraging the norm of the activations to be small reminds the weight decay regularisation, where the weights are encouraged to be small. The relationship between the spatial smoothness concept and L2 regularisation can be analysed in the simple case where the filtering operation returns the raw activations:

$$\tilde{\boldsymbol{H}}_t^l = \boldsymbol{H}_t^l \tag{4.17}$$

From the definition of Frobenious norm:

$$||A||_F = \sqrt{\sum_i \sum_j a_{ij}^2}$$

R^{Smooth} as:

$$R^{Smooth}(\theta) = \sum_l \sum_i h_i^{(l)2} \tag{4.18}$$

Under the assumption that the individual activations $h_i^{(l)}$ are obtained by a *tanh* function (which we will call with $\psi(\cdot)$ for brevity) :

$$h_i^{(l)2} = \psi^2(\sum_j w_{ij}^{(l)} h_j^{(l-1)}) \tag{4.19}$$

The *tanh* can be approximated by a Taylor series:

$$h_i^{(l)2} \approx \left((\psi(0) + \psi'(0) \sum_j w_{ij}^{(l)} h_j^{(l-1)} \right)^2 \tag{4.20}$$

$$= K \left(\sum_j w_{ij}^{(l)} h_j^{(l-1)} \right)^2$$

where $K = \psi'(0)$ and $\psi(0) = 0$. Rewriting Eq. 4.20, where J is a constant:

$$h_i^{(l)2} \approx KJ^2 \left(\frac{1}{J} \sum_j w_{ij}^{(l)} h_j^{(l-1)} \right)^2 \tag{4.21}$$

The application of the *arithmetic mean-geometric mean inequality* leads to:

$$h_i^{(l)2} \leq KJ \sum_j w_{ij}^{(l)2} h_j^{(l-1)2} \tag{4.22}$$

The sum in Eq.4.22 is the *arithmetic-harmonic mean*, which allows us to rewrite the equation as:

$$h_i^{(l)2} \leq KJH \sum_j w_{ij}^{(l)2} \tag{4.23}$$

$$h_i^{(l)2} \leq C \sum_j w_{ij}^{(l)2} \tag{4.24}$$

where C has been used as a cumulative constant to gather the terms brought out from the sum. The total regularisation is given by the sum overall layers, which leads to:

$$R^{Smooth}(\theta) \leq C \sum_l \sum_i \sum_j w_{ij}^{(l)2}$$

Hence, L2 is an upper bound for the spatial smoothness regularisation.

The non-filtered constraint indirectly acts on the weights as the L2 regularisation, encouraging them to remain small. While L2 acts on the full weight matrices, spatial

smoothing is only applied on the activations of the hidden layer. For example, if both the embedding layer and hidden layer have the same number N of hidden units, then the L2 regularisation has a computational cost of $O(N^2)$, while spatial smoothing has cost of $O(N)$. Therefore, spatial smoothing could be an efficient regularisation term in case of very large networks.

This would result extremely beneficial in case of the training of very large networks, since it would avoid the overfitting on the training data at a lower cost than the L2 regularisation.

Chapter 5

Network Interpretability

Several approaches have been used to to obtain a meaningful representation of neural networks activations, especially for Convolutional Neural Networks [30, 12, 54, 17]. The tradional methods invert the transformations inside the network use the signals from the forward propagation to reconstruct the features detected by each neuron. While the interepretation is almost immediate for digital image inputs, in case of sequential inputs interpretability is generally limited to the individual analysis of ranges of the activations, their saturation points, etc., which may result cumbersome and not straightly intuitive.

In the next sections we illustrate the main issues connected to the interpretation of sequential data inputs and we propose two different methods to convey meaning to the network hidden activations.

5.1 Interpretability for sequential data inputs

Interpreting sequential data inputs can be less intuitive than interpreting images, especially when they are multidimensional. The analysis of multidimensional data through different visualisation methods has been addressed since the early twentieth century, when scientists such as Pearson and Fisher developed methods for the analysis of multivariate data [29].

In case of speech signals, Sensitivity-Characterised Activity Neurograms have been proposed by Tan et al. to analyse the activations of a deep feed forward network [44]. Alternatively, the stimulated training of DNN proposed an interpretable training of an acoustic model, where a visible pattern moves in 2D map of phonemes as the input is observed[39, 50] .

5.2 Visualisation of the recurrent layer

The qualitative interpretation of the embedding process of recurrent units such as LSTMs cells and GRUs has recently become of interest [54, 5, 27, 37, 45] and allows to investigate important properties of the gated units. For instance, Palangi et al. visually showed the LSTM ability to capture long term memory dependencies and forget irrelevant information by automatically attenuating unimportant words and detecting the salient keywords in sentences.

Furthermore, Karpathy et al. analysis of the LSTM cell showed that LSTM can robustly identify interpretable high-level patterns such as lines, lengths, brackets and quotes. Such properties allow to treat sentences as sequences of words with internal structures, yielding at the end of the encoding process a cumulative representation of the whole sentence [37].

5.3 Activation images

The first issue for interpretability is the visualisation of the activations. The hidden layer should follow a spatial organisation easy to visualise and meaningful, so that interpretation is possible. For instance, the activation vector can be reorganised in a 2D-flat grid as proposed in Eq. 4.2, yielding an activation image where the activation intensities in the different regions of the space are rendered with different colours.

If just a random reorganisation is selected, activation images are similar to the one shown on the left in Figure 5.1, where the activations do not follow any meaningful ordering. Promoting a similar behaviour of nearby activations would allow visible activation patterns to form, similar to the one proposed on the right in Figure 5.1. Each activation pattern could specialise on one feature of the input. For example, lets imagine that for the input word 'OKAY' a region of the activation matrix constantly activates. We would like to observe a similar pattern for 'OK' , since essentially the two words correspond to the same concept.

Conveying meaning to the activation image The definition of a meaningful mapping within the activation image would allow to convey a meaningful interpretation of the activation patterns. If we introduce a mapping of the space where each region represents a specific feature of the input, then we can inject interpretability to the representation of the activations. Moreover, having different regions being active depending on the input unit may help the network to discriminate better and as a

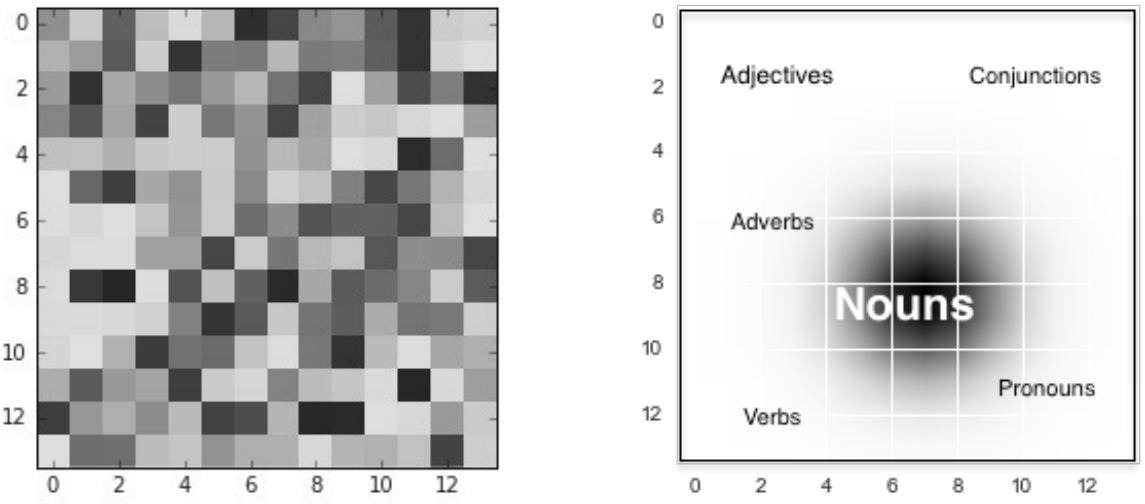


Fig. 5.1 *Comparison between two reorderings of the activation patterns. Left:* random reordering. *Right:* desired behaviour of the activations in an interpretable handcrafted map.

consequence yield lower error rates[39]. For example, Figure 5.1 compares a random ordering of the activations to a preferable ordering, which defines different patterns according to the syntactic structure of English sentences.

5.4 Exploitation of target patterns

The identification of a target pattern for the internal activations of the network is strictly connected to the external meaning that we would like to inject in the network training. For instance, it could be useful to incorporate in the mapping the semantic and the grammatical relationships that may be found in a sentence. For this reason, in the following sections we present two different target patterns: the first uses directly the *Word2vec* embeddings, which proved successful in expressing the semantic closeness between words; the second uses Part-of-Speech tagging to include syntactic information in the mapping.

5.4.1 *Word2vec* embeddings

Several work has been done recently to learn real-valued vector representation of words [19, 34, 36]. Mikolov et al. Skip-gram model is an efficient method for learning high-quality distributed vector representations of words, generally known as *word2vec*. In this representation the syntactic and semantic word relationships are encoded in

such a way that words that occur in similar contexts are closer in the learned vector space. For instance, common examples are the vector operations:

$$v(Paris) - v(France) + v(Italy) \rightarrow (Rome)$$

$$v(king) - v(man) + v(woman) \rightarrow v(queen)$$

where $v(\cdot)$ returns a vectorial word2vec representation, and $\rightarrow$ symbolises the closest word2vec vector to the one obtained [34, 19].

The word2vec space constitutes a meaningful space to map words, since close words in this space generally occur close in natural language. The Google billion corpus has been used as the training corpus to obtain the projections.

Google billion corpus The Google billion words corpus is a benchmark with over one billion words of training data [1], which has been used to evaluate novel language modeling techniques and to compare their contribution when combined with other advanced techniques [6]. The data were gathered by monolingual English corpora and duplicated sentences were removed. The vocabulary includes most of the existent English words, consisting of 793 471 individual words.

The pretrained Google billion *word2vecs* are opened to the public, and consists 300-dimensional vectors which represent each of the words in the corpus. Such large number of word vectors can be used to obtain a dense 2D space where most of the word similarities are mantained in the projection. Generally, dimensionality reduction methods such as T-sne [32] or Principal Components Analysis are used. Figure 5.2 shows a portion of the word2vec space for the Google billion corpus embeddings. The cluttered nature of the projection space is useful to represent semantic similarities, although it makes interpretability extremely complicated.

The regularisation term is obtained by placing Gaussian targets on top of each projected word vector and computing the cosine distance between the word embedding in the neural model and the target embedding in the word2vec target map.

5.4.2 Part of speech target maps

Part of speech patterns can be used to encourage the grid regions to model the syntactic structure of the sentences. A Part-Of-Speech Tagger is an algorithm that

[1] The benchmark is available as a (code.google.com project)

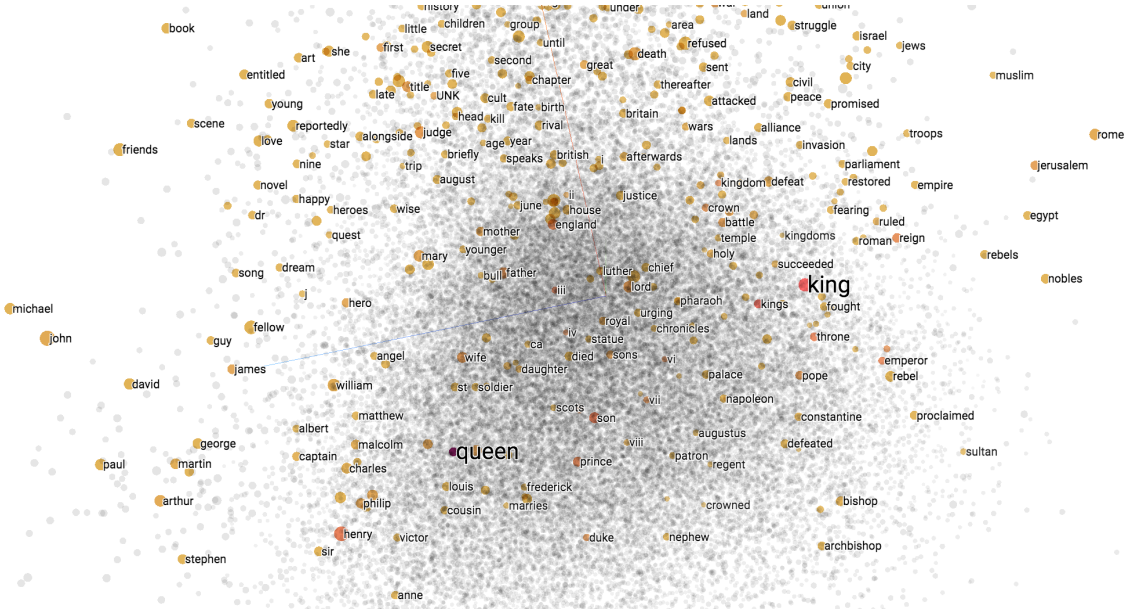


Fig. 5.2 *3D Tensorboard [1] visualisation of the PCA of a subset of the Google bilion Word2vec Embeddings.*

reads a textual input and assigns to each token a part of speech label, such as noun, verb, adjective, etc [47]. Part of speech categories group together words which have similar grammatical properties and similar behavior in terms of syntax or morphology. Ideally, we would like to obtain a map that encodes the syntactical structure of English sentences. For example, we would like to have the the nouns grouped together and close in space to verbs and articles, since even the simplest sentence is generally composed by an article, a noun and a verb.

POS Regularisation The POS regularisation term can be obtained following the same procedure explained in [50] for acoustic features. The *word2vec* vectors belonging to each POS category are averaged to obtain the centroid of each category. Dimensionality reduction methods such as T-SNE [32]are used to project the 300-dimensional centroids on a 2D map, which should enhance the natural smooth transitions between similar contexts. Since T-SNE is very dependent on the initialisation of some parameters (i.e. initial perplexity, number of iterations, etc.), Multidimensional Scaling (MDS) [31] can be used as a valid alternative. MDS maintains information on the original distances between two elements in the high-dimensional space, while evidencing as much as possible similarities and differences between group of elements.
Once identified the mapping, a normalised Gaussian distribution is placed on top of each category point and the final regularisation term is obtained by minimising the

KL-divergence between the activations distribution and the target distribution as in Eq. 4.10.

Chapter 6

Experiments

6.1 Training data

Two different training sources, namely the AMI meeting corpus and the Multi Genre Broadcast Challenge (MGB) were used. Table 6.1 presents summary statistics on the data.

Dataset	Vocsize (W)	(train) Size (MW)	(val) Size (W)	(test) Size (W)
AMI meetings	32K	1.0	121 K	115 K
MGB3 transcription	43K	4.8	71 K	-

Table 6.1 Corpus statistics

6.1.1 AMI meeting transcriptions

The AMI Meeting Corpus is a public set of recorded meetings which has been collected by The European-funded AMI project. The training set consists of 78 hours of meeting recordings for which high quality, manually produced orthographic transcription for each individual speaker are provided. Eight meetings were kept from the training set and used as development and test sets. The corpus consists of approximatively 100 thousand sentences for a total of 1M words and approximatively 32K unique words in the vocabulary.

6.1.2 Multi-Genre Broadcast Challenge

The MGB Challenge is an evaluation of speech recognition, speaker diarization, dialect detection and lightly supervised alignment using TV recordings supplied by the British Broadcasting Corporation (BBC) and consists of audio from BBC television

system	dev PPL	test PPL
3-gram	93.6	82.8
CUED-RNNLM v1.0 - LSTMs	79.91	71.7
TF-RNNLM-AMI - LSTMs	76.23 ± 0.30	69.10 ± 0.23

Table 6.2 *Baseline system performances on AMI.* The TF-RNNLM-AMI results are averaged on three different reinitialisations.

programmes. The data is very varied and covers the full range of genres (e.g. comedy, drama, sports shows, quiz shows, documentaries, news etc). The speech data is broad and multi-genre, spanning the whole range of TV output, and represents a challenging task for speech technology. The corpus consists of approximatively 4.8 M words, 63K unique words, which have been short-listed to the most frequent 42K.

6.2 Network Configurations

The basic RNNLM structure used for the experiments [1] comprehended input and output layers over the full vocabulary, a projection layer and a recurrent layer with either LSTMs and GRUs cells. An out-of-vocabulary node has been used to represent any input word not in the recognition vocabulary. Training used the BPTT algorithm, where the error is propagated back through the recurrent connections for 20 time steps. The initial network weights were set with the Xavier initialisation and the initial learning rate tuned between 1e-2 and 1. The Adam optimiser has been used with gradient clipping to a maximum norm of 10.
This configuration sightly differs from the CUED-RNNLM configuration, which uses the BPTT algorithm for 5 timesteps, an out-of-shortlist node, mini-batch SGD optimisation and initial learning rate of 1.0.

TF-RNNLM-AMI Nearly 43K input and output nodes are used in TF-RNNLM-AMI. The embedding projection layer has 196 units and the recurrent layer uses 196 LSTM cells. The system performance has been compared to the CUED-RNNLM baseline in Table 6.2. The TF-RNNLM-AMI PPL reaches on average over a set of three random reinitialisations 76.23 PPL on the development set and goes down to 69.10 on the test set. This results are slightly lower than the CUED-RNNLM baseline, which is probably due to the small differences in the two model configurations. However, the obtained results are still comparable with the CUED-RNNLM baselines.

[1] Credits to A. Malinin for the base model Tensorflow implementation

6.3 Experiments on Spatial smoothing

In this section we present the experiments on the spatial smoothing regularisation.

The spatial smoothing regularisation term is obtained by following the procedural steps presented in Section 4.2. The hidden units activations are organised in a 14x14 flat grid and their convolution with a 3x3 box filter kernel $\boldsymbol{K}$ is used to obain the high-pass filtered activation grid, $\tilde{\boldsymbol{H}}_t^{(l)}$. A static target pattern $\boldsymbol{G}^{(l)} = 0$ is fixed. The MSE criterion is used to compute the distance between the transformed activations and the target pattern. Figure 6.10 shows the learning curves of the models, which to converge on average after 15 epochs.

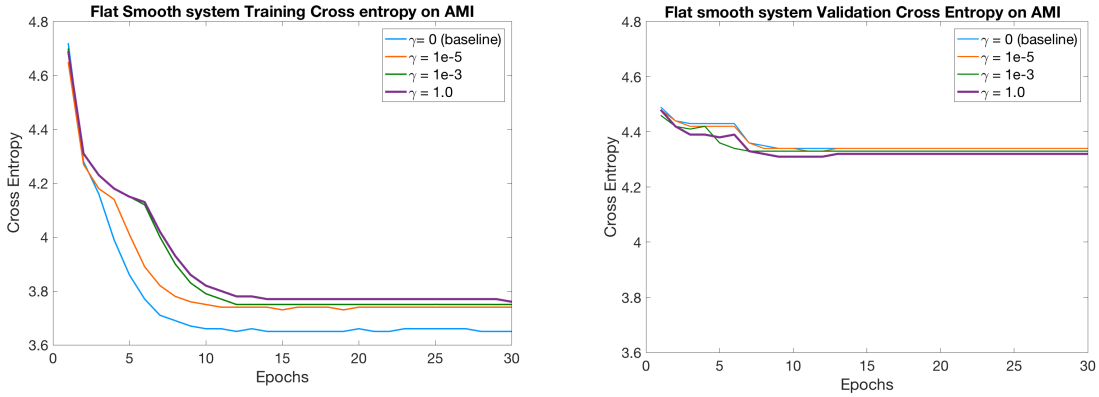


Fig. 6.1 *Flat Smooth system training and validation curves for different regularisation penalties on the AMI dataset.* The light blue curve represents the baseline TF-RNNLM model, where no stimulation has been applied.

Experiments Spatial smoothing has been investigated on both models with a flat grid reorganisation of the activation (which will be called Flat Smooth systems) and with toroidal surfaces (e.g. Toroidal Smooth systems). Experiments cover the investigation of a various range of model properties with the final goal of finding the best smoothing model configuration. First, we investigated the impact of the regularisation penalty on the dynamic range of the regularisation. Second, we compared the flat organisation of the activations to the toroidal organisation and we investigated the use of either stronger or bigger filter kernels. Modification of the grid sizes were analysed to see how the regularisation scales on larger models. Finally, models with dropout were trained and compared to the smoothing models.

Impact of the regularisation penalty

The regularisation penalty γ controls the strength of the regularisation term in the training objective function. Larger values of γ correspond in final higher training costs and in better validation performances, as shown in Figure 6.2. For instance, for increasing values of gamma the training cross entropy raises from 3.68 to 3.78, whereas the validation error decreases as γ increases, reaching the minimum of 4.30 for $\gamma = 1e-2$.

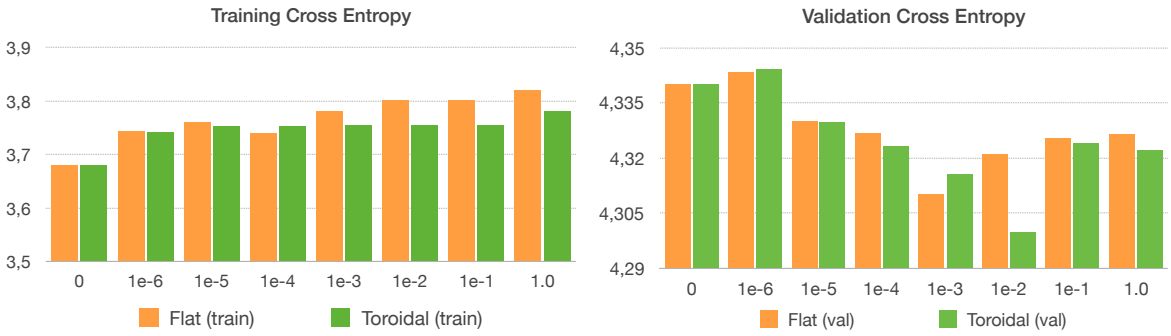


Fig. 6.2 *Final training and validation cross entropy costs for different regularisation penalties on AMI.*

Figure 6.3 shows the trade-off between the regularisation and the cross entropy cost for different values of the regularisation penalty.

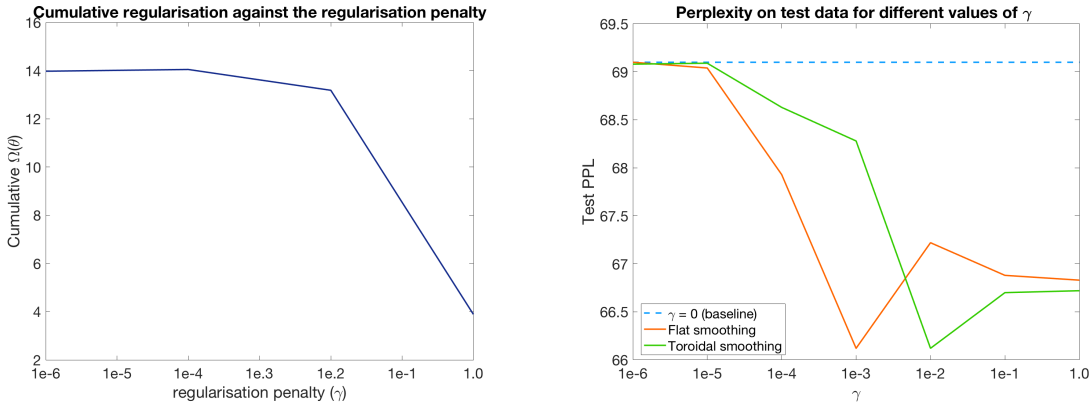


Fig. 6.3 *Tuning of the regularisation penalty. Left:* Cumulative regularisation (MSE) added during training for different values of the regularisation penalty γ on a log scale. *Right:* Perplexity on the testing set against regularisation penalty.

Higher values of γ lead to lower values of the cumulative regularisation term added during the training process. For instance, when $\gamma = 1.0$ the order of magnitude of $R^{Smooth}(\theta)$ reaches is 10^1, whereas for $\gamma = 1e-5$ it ranges between 10^5 and 10^6.

Hence, larger regularisation penalties encourage smoother activation surfaces, leading to smaller cumulative regularisation terms.
The oscillation of the perplexity on held out test data on the right in Figure 6.3 shows that the best trade-off compromise is given by $\gamma = 1e-3$ for the Flat Smooth system and by $\gamma = 1e-2$ for the Toroidal Smooth system.

Introduction of the toroidal curvature

The organisation of the hidden nodes in a torus leads to the closed surface in Figure 6.4. The continuous toroidal shape allows to avoid the filtering edge-effects at the borders of the activation image.

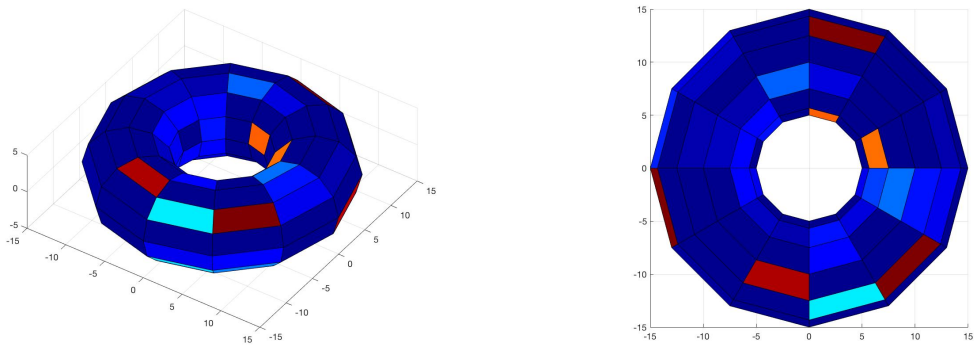


Fig. 6.4 *Network activations on a toroidal organisation of the hidden units*

The tuning of the regularisation penalty on the Toroidal Smooth system followed the same considerations made on the Flat Smooth system. For instance, the perplexity oscillation of the toroidal model for different penalties is shown on the right in Figure 6.3 (in green). The PPL of the model decreases as more and more importance is given to the regularisation penalty. The model with $\gamma = 1e-2$ yields also the lowest values of the validation cross entropy, at 4.30 against the baseline value of 4.34.

Generalisation to unseen data Table 6.3 shows the generalisation performances of the best systems on held out test data.

Both the smoothing systems yielded consistent improvements in the development and test perplexities with respect to the baseline model. The toroidal smooth system yields the best average performances over a set of 3 repetitions, with an average decrease of 3.0 points in the test PPL. The introduction of a short-list for the output word probabilities of the most $\approx$ 22K used words in the vocabulary (as proposed in

system	γ	dev PPL	test PPL
TF-RNNLM-LSTM	0.0	76.23 ± 0.30	69.10 ± 0.23
Flat Smooth-LSTM	1e-3	74.47 ± 0.40	66.20 ± 0.42
Toroidal Smooth-LSTM	1e-3	**74.01** ± 0.35	**66.12** ± 0.32

Table 6.3 *Smooth LSTM system performances over different initialisations on AMI.*

[8]) yields a further average improvement of 2.0 points in the development perplexity (e.g. development perplexity 72.00 and test perplexity 66.15), which could be further enhanced by the introduction of an out-of-shortlist output node. The use of GRU did not lead to better performances, although toroidal smoothing decreases the test PPL of 3.0 points even in GRU models, reducing the test PPL of the baseline model GRU from 75.46 to 72.36 (see Table 6.4).

system	γ	test PPL
TF-RNNLM-GRU	0.0	74.46 ± 0.24
Toroidal Smooth-GRU	1e-3	**72.36** ± 0.31

Table 6.4 *Smooth GRU system performances over different initialisations on AMI.*

Analysis of different filter kernels In addition to the 3x3 box filter, the application of a 5x5 linear filter kernel and of a 5x5 Gaussian high-pass filter (in Figure 6.6) has been investigated. Figure 6.7 shows the relative training and validation learning curves of the Toroidal Smooth system. Overall, the 3x3 kernel still gives the best performance when compared to bigger filters. Bigger filters or more powerful filters such as the Gaussian high-pass filter (in Figure 6.6) introduce a stronger regularisation, which causes the models to underfit the training data, as can be seen in Figure 6.7. Larger networks with more than only 196 units could benefit from a regularisation with a larger filter, since the stronger regularisation could avoid overfitting on the training data.

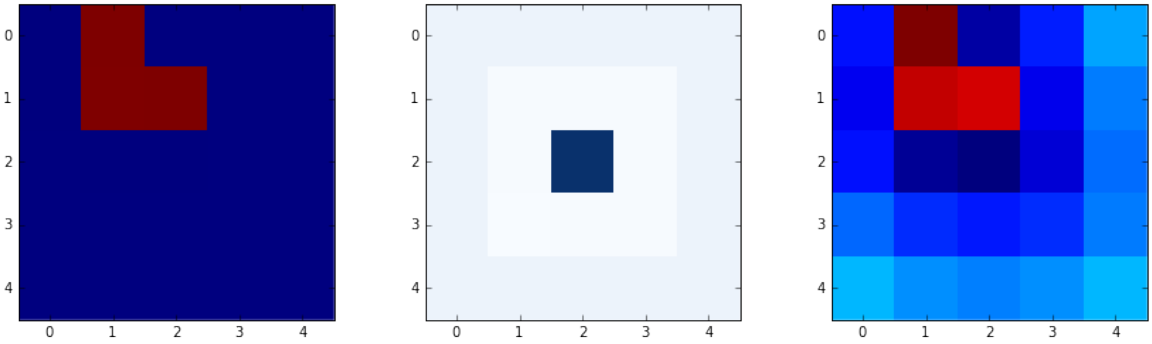


Fig. 6.5 Application of a 5x5 linear smoothing kernel (1 in the central box , -0.08 in the immediate surroundings and -0.02 in the furthest part [5]) on a sample pattern. *Left:* activations sample pattern. *Centre*: filter *Right*: filtering output.

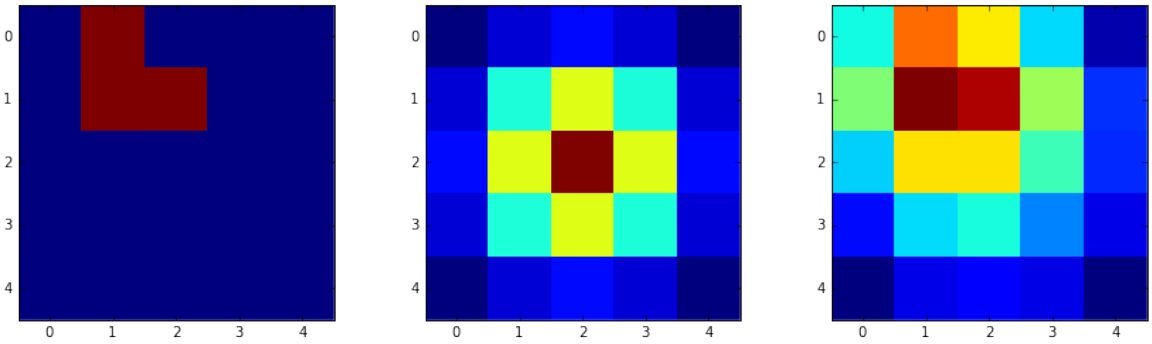


Fig. 6.6 *Application of a Gaussian high pass filter on a sample pattern. Left:* activations sample pattern. *Centre*: filter visualisation *Right*: filtering output.

Analysis of different sizes of the hidden layer The impact of the regularisation on bigger networks has been tested using a larger model, with 1024 LSTM cells. Figure

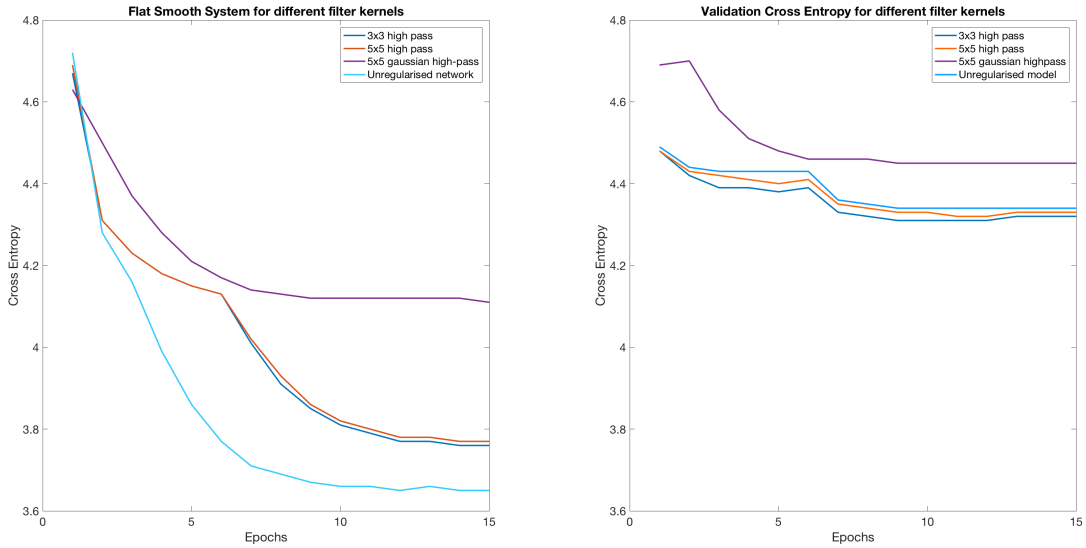


Fig. 6.7 *Flat smooth system with different filter kernels. Left:* training Cross Entropy. *Right:* validation Cross Entropy. The gaussian filter causes underfitting.

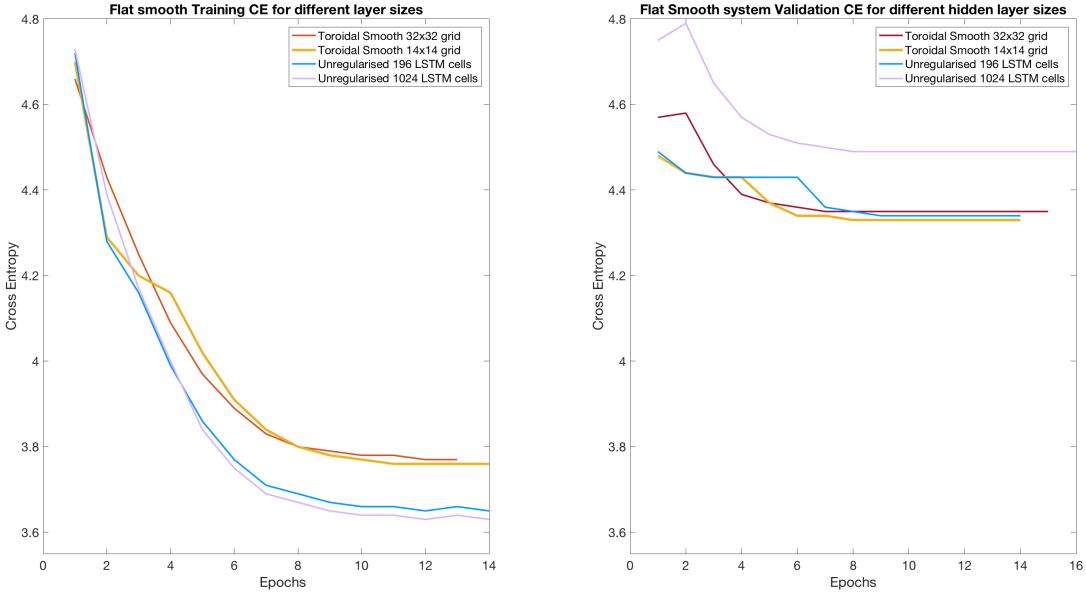


Fig. 6.8 *Regularisation of larger hidden layers compared to the unregularised corresponding networks.* Overfitting is prevented by the introduction of the regularisation term.

6.8 shows how the regularisation introduced by spatial smoothing avoids overfitting. For instance, the validation cross entropy of the toroidal smooth system with 1024

units converges to the same range of values of the smaller model. The regularisation penalty was tuned on the bigger model and set at $1e-4$. Normalising the regularisation term over the number of hidden units in the layer could remove the dependency of the regularisation penalty on the layer size.

Comparison between spatial smoothness and dropout Table 6.5 shows the comparison between the perplexity on the validation set and on held-out test data for the toroidal smoothing model and a standard RNNLM with dropout. The development perplexity is really high in the dropout models, but they have strong generalisation performances. The generalisation performances of the toroidal smooth model is comparable to the performances of the RNNLM with dropout.

System	dev PPL	test PPL
Dropout (p=0.5)	87.83 ± 0.22	66.63 ± 0.31
Dropout (p=0.8)	81.19 ± 0.18	67.54 ± 0.32
Toroidal Smooth	74.47 ± 0.40	**66.12** ± 0.32

Table 6.5 *Comparison of development and test PPL values of the TF-RNNLM with toroidal smoothing and with dropout .*

ASR performances Table 6.6 reports the WER evaluation for the best analysed models. The Toroidal Smoothing system reports slights WER improvements. The WER has been computed rescoring the 50-bests list of the ASR output of a Kaldi acoustic model training recipe featuring sequence training [48, 9] with FMLLR [2] transformed MFCC features [3] [40, 13]. The RNNLMs has been interpolated with a 3-gram model, with language model penalty tuned between 10 and 12. The LM interpolation weights have been manually tuned between 0.5 and 0.8 for the 3-gram model and 0.5 and 0.2 for the Tensorflow models. The difference between the CUED-RNNLM WER and the TF-RNNLMs WERs underlines that the performances of the two models are not directly comparable. This is probably due to the structural differences between the two models, as explained in Section 6.2. However, improvements in the WERs of the Tensorflow models are reliable since the Tensorflow models configurations are consistent.

[2]FMLLR:Feature space Maximum Likelihood Linear Regression
[3]MFCC: Mel-frequency cepstral coefficients

system	+WERs
3-gram	25.8
CUED-RNNLM v1.0 - LSTMs	25.3
TF-RNNLM-AMI - LSTM	24.6
Flat Smooth-LSTM	24.5
Toroidal Smooth-LSTM	**24.5**

Table 6.6 *ASR performances on AMI with LM rescoring.*

6.4 Experiments on the MGB dataset

TF-RNNLM with toroidal smoothing and GRU units were trained on the MGB dataset to apply the framework on a different corpus. MGB has in fact very different features from AMI, since it is a much larger corpus and contains varied language styles.

TF-RNNLM-MGB In the TF-RNNLM-MGB model 196 GRUs were preferred to LSTMs to reduce the computational complexity and a shortlist was used to limit both the input and the output layer to the most 43K frequent words. The bias to in-shortlist words during RNNLM training could be reduced with an additional out-of-shortlist node at the output layer.

Toroidal smoothing The toroidal smoothing system has been applied to the three different presented models. Table 6.7 reports the final perplexities of the models. As the size of the hidden layer increases, larger unregularised networks tend to overfit the training data. The smooth regularisation allows the training of very large models without any decrease in performances due to overfitting.

	14x14	25x25	30x30
unregularised	182.79	199.60	204.10
toroidal smooth	179.11	179.76	**178.52**

Table 6.7 *MGB validation perplexity of the unstimulated and the stimulated models (TF-RNNLM-GRU).*

Further experimentation on the MGB models could have been carried out to investigate the impact of the regularisation penalty and of the use of different filters. However the long training time of the models did not allow to expand the investigation further during the project time period.

6.5 Experiments on Interpretability

Related work on stimulated learning [39, 50, 44] showed that the injection of external knowledge during the training process could lead to the formation of interpretable patterns in the hidden activations as well as yielding better generalisation. Interpretability has been investigated on the AMI corpus, by looking at two different types of target patterns. First, the Skipgram model *word2vec* embeddings of the Google billion word corpus have been used in a 'raw target map' (e.g. raw-map system). Second, syntactic information has been included using Part-of-Speech tagging labels (e.g. POS-MDS system).

Network configurations The same network configurations in Section 6.2 have been used, and both *tanh* and sigmoids have been investigated with the cosine distances. The KL divergence regularisation has been used only with sigmoids as suggested in [50]. The learning curves and the tuning of the regularisation penalty γ are reported in Figure 6.10.

Raw-map targets The *word2vec* embeddings of the AMI vocabulary are obtained from the online pre-trained models with *gensim* [4] [36] . Each embedding is computed with the contextual information about the surrounding words, and therefore the *word2vec* embeddings are likely to represent both the semantic and syntactic features of the words [36]. The embeddings are then projected on a 2-dimensional grid with T-SNE [32] and Gaussian targets with 0.5 standard deviation are centred on each of the projected word points (see Figure 6.9).

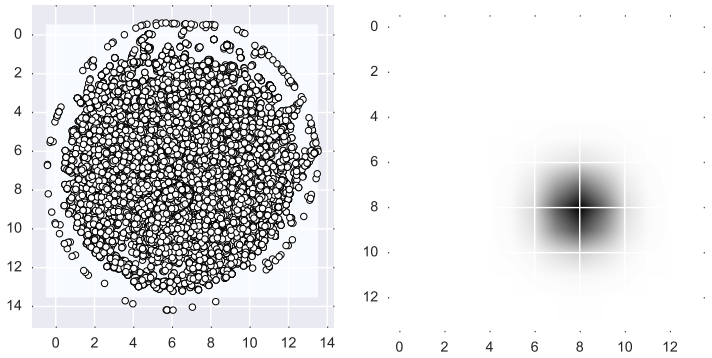


Fig. 6.9 *T-SNE word embeddings targets. Left:* T-sne projection of the word embeddings on the 2-d grid. *Right:* gaussian target pattern for 'OKAY'.

[4] https://radimrehurek.com/gensim/

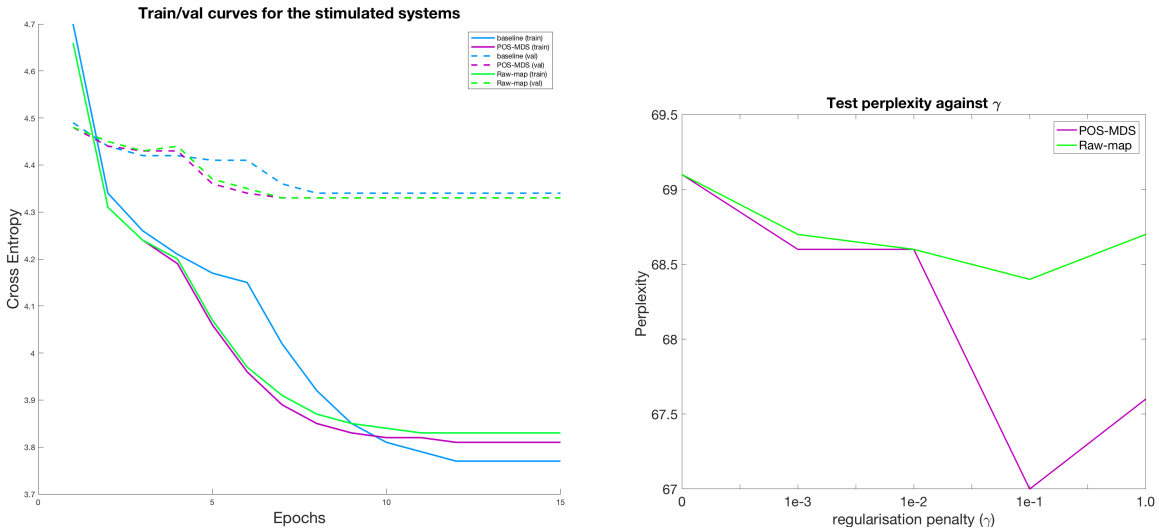


Fig. 6.10 *System training learning curves and parameters tuning. Left:* raw-map and POS-MDS system training and validation curves as opposed to the baseline RNNLM (where sigmoids have been used instead of tanh) *Right:* Tuning of the regularisation penalty γ

Response to one input sentence The raw *word2vec* stimulation shows that the gating structure in the LSTM cells keeps track of the previous activation patterns throughout the sentence. For example, Figure 6.11 shows the *'activation canvas'* response to the input sentence 'HERE WE GO'. As the words pass through the network different areas are stimulated while some of areas remain active. At the end of the sentence the activation grid can be seen as an 'embedding' of the full sentence [37].

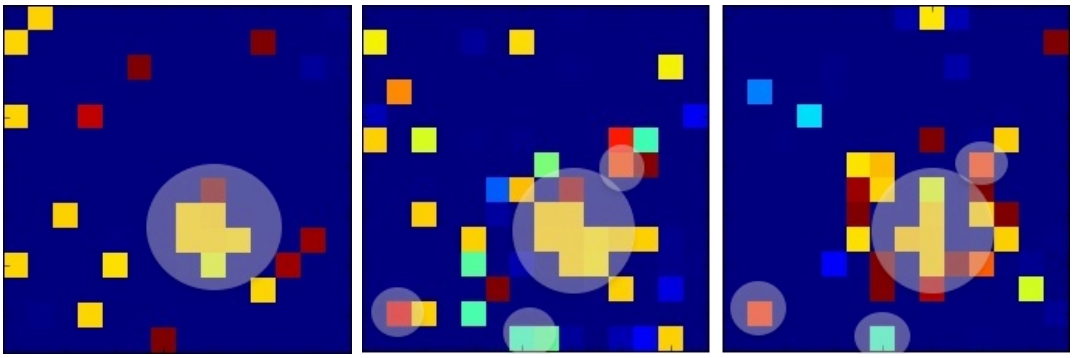


Fig. 6.11 *Raw word2vec stimulation response with the sentence input 'HERE WE GO'. The units kept in the LSTM memory have been evidenced.*

The stimulation leads to nearly 2% improvement points in the test perplexity of the model (see Table 6.8), showing that the incorporation of external information helps the

training process. A further tuning of the standard deviation of the Gaussian targets could lead to better performances. Notwithstanding, the cluttered nature of the raw *word2vec* projections makes interpretability stil challenging.

System	γ	dev PPL	test PPL
TF-RNNLM	0	75.69	69.10
Raw-map	1e-1	75.00	67.20
POS-MDS	1e-1	74.84	68.20

Table 6.8 *Development and Test PPL for each of the different systems.*

***Semantic* targets** The second experiment involved the introduction of semantic structure in the activation targets. The Stanford Natural Language Parser [5] [11] has then been used to assign POS tags for each of the Skipgram *word2vec* embeddings of the Google billion corpus. The average word embeddings are computed for each POS category, obtaining an indicative POS embedding. The POS embeddings are reduced with Multi Dimensional Scaling (MDS) from 300 dimensions to just 2, and subsequently centred and scaled to fit the grid size. Figure 6.12 shows the POS target map obtained by projecting the average embeddings for each POS category on the 2d-grid. The POS tags descriptions are reported in the Appendix.

Ideally, the target map should automatically encode meaningful characteristics of the input data. The traditional structure of English sentences is reflected in the POS-MDS map, where 'similar' POS tags are close together. For instance, the syntactic structure of the sentence is encoded through the closeness of the tags for determiners (DT), nouns (NN and NNs) and verbs (VB), which generally are close to each other and represent the key elements of English sentences.

The stimulated training followed the procedural steps reported in [50]. The target pattern and the activation grid are first converted into probability mass functions. The regularisation term is then computed by looking at the KL-divergence between the target distribution and the current activation distribution. Figure 6.13 shows an example of the activation patterns for the input word 'OKAY' in the baseline system, the Toroidal smooth system and the POS-MDS system. The first three images refer to systems with *tanh* activations, hence the green color corresponds to zero, blue to -1

[5]https://nlp.stanford.edu/software/lex-parser.shtml

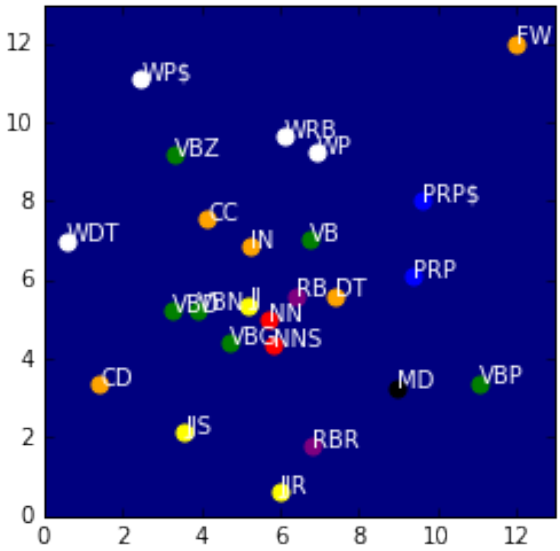
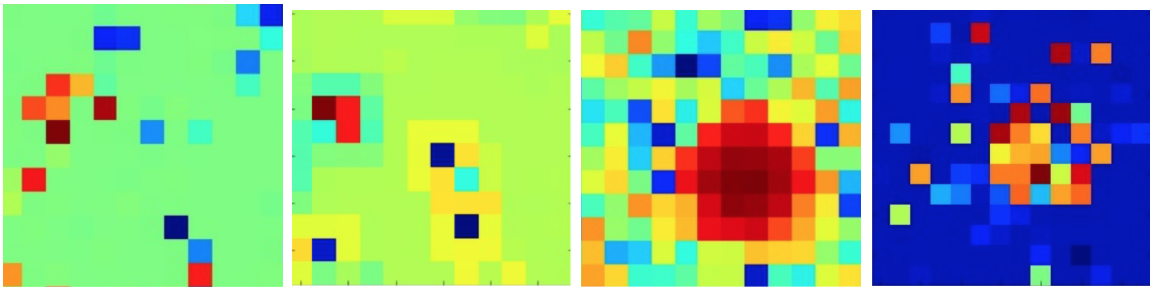


Fig. 6.12 *POS targets map*

and red to +1. The last image has been obtained from a system with sigmoids, hence blue corresponds to zero and red to one. Both the effects on smoothing and of the target specific stimuly are noticeable in the output images.



(a) (b) (c) (d)

Fig. 6.13 *Internal Representation of 'OKAY' (RB, Adverb) in the different Systems. (a): Unregularised network, tanh activations. (b): Toroidal Smoothing, tanh activations. (c): Raw-map target, tanh activations. (d): POS-MDS target, sigmoid activations*

System performances The application of POS-MDS stimulation only brings slight improvements in the testing perplexity of the system, e.g. 0.9%. This might be due to different factors that should be more carefully addressed in the development of the POS stimulated system. To begin with, the labeling of the POS tagger may introduce a certain degree of error. For instance, the tagger addresses each word individually and not directly in the specific context. The introduction of sentence-specific tags

(i.e. tailored on the training corpus) would address this problem, thus reducing the context shift between the Google billion word corpus and the AMI corpus. Moreover, a proper tuning of the standard deviation σ of the gaussian targets is needed to ensure better performances. If σ is too large, the target may cover more than one POS cluster, whereas if σ is too small, the activations are encouraged to behave as delta inputs on the activation grids, thus removing all the benefits given by a smooth activation surface.

The word patterns present two main differences from the acoustic patterns in [50, 39]. The acoustic patterns were more clearly delineated and the change of pattern was characterised by a smooth transitions between the previous and the next pattern. The first difference may be addressed by using larger grids and stronger regularisation terms. The absence of smooth transitions may be intrinsic of the language modelling problem and due to to the discrete nature of language. For instance, the acoustic sounds are characterised by continuous transitions between one sound and the other, which are well captured by the stimulation. Conversely, language does not present smooth transitions between words. Both the raw-map and the POS-MDS regularisation can be better interpreted as the imposition of target-specific smoothing constraints on the activation surface. The gaussian target enhances the activation of a particular region of the activation grid (see Figure 6.13) while driving to zero the activations outside of the target area. In both mappings the targets with similar features (either semantic for *word2vec* or syntactic for the POS maps) are close to each other, and therefore the activations related to similar features fall under the gaussian target, which encourages them to form a smooth surface.

Chapter 7

Discussion and Future Work

In this project we addressed interpretability and generalisation, which are two major issues in deep learning. Especially, we focused on the conceptual question of whether it is possible to bring improvements in both the training process and the generalisation capabilities of RNNLMs by defining alternative regularisation terms. A regularisation term that promotes the formation of smooth surfaces in a toroidal reorganisation of the recurrent layer activations has been proposed and compared to the smoothing of flat activation grids. Experimental results demonstrate that the toroidal smoothness stimuli affects the training criteria, yielding to better generalisation performances.

The results are consistent with the existing research on Stimulated Learning. The same framework principles which have been previously used on ASR tasks have been presented on a language modeling task for the first time. Similarly, Recurrent architectures have been used for the first time instead of multilayer feed forward networks. The conventional spatial smoothing system has then been explored on a novel organisation of the activations on a closed toroidal surface. Experiments on the interpretability task have used simple off-the-shelf methodologies to investigate possible ways of injecting external knowledge on the syntactic structure of English sentences in the network training criterion.

Overall, the spatial smoothing regularisation yields better performances than the target specific stimuli. Moreover, the smoothed systems allow the training of larger networks and prevents overfitting, and yielded comparable results to the application of dropout. This lets believe that spatial smoothing could be a valid alternative to the common regularisation terms.

More research in this area is still necessary before claiming the statistical significance of this results. As first instance, the application of the RNNLM should be conducted on different language modeling tasks, especially to verify how the regularisation term scales with the dimensionality of the training corpus. For instance, there seems to be a

relationship between the size and strength of the filter used and the size of the network, which could not be further addressed in this work because of time limitations. Moreover, the ability of spatial smoothing to prevent overfitting in case of large networks should be verified on even larger architectures, e.g. 4096 nodes and even more.

The analysis on interpretability should be further expanded by the definition of better mappings, for instance starting from a statistical POS parser, which allows the assignment of the tags with some degree of uncertainty. Moreover, the parsing should be done directly on the training data at the sentence level in such a way to reduce the errors introduced by the parser because of language ambiguities.

The further investigation of these areas could lead to even better improvements in the network interpretability and generalisation capacities.

References

[1] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., et al. (2016). Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*.

[2] Becker, S., Le Cun, Y., et al. (1988). Improving the convergence of back-propagation learning with second order methods. In *Proceedings of the 1988 connectionist models summer school*, pages 29–37.

[3] Bengio, Y., Ducharme, R., Vincent, P., and Jauvin, C. (2003). A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155.

[4] Bishop, C. M. (1995). *Neural networks for pattern recognition*. Oxford university press.

[5] Burger, W., Burge, M. J., Burge, M. J., and Burge, M. J. (2009). *Principles of digital image processing*. Springer.

[6] Chelba, C., Mikolov, T., Schuster, M., Ge, Q., Brants, T., Koehn, P., and Robinson, T. (2013). One billion word benchmark for measuring progress in statistical language modeling. *arXiv preprint arXiv:1312.3005*.

[7] Chen, S. F. and Goodman, J. (1996). An empirical study of smoothing techniques for language modeling. In *Proceedings of the 34th annual meeting on Association for Computational Linguistics*, pages 310–318. Association for Computational Linguistics.

[8] Chen, X., Liu, X., Qian, Y., Gales, M., and Woodland, P. C. (2016a). Cued-rnnlm—an open-source toolkit for efficient training and evaluation of recurrent neural network language models. In *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*, pages 6000–6004. IEEE.

[9] Chen, X., Liu, X., Wang, Y., Gales, M. J., and Woodland, P. C. (2016b). Efficient training and evaluation of recurrent neural network language models for automatic speech recognition. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 24(11):2146–2157.

[10] Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

[11] De Marneffe, M.-C., MacCartney, B., Manning, C. D., et al. (2006). Generating typed dependency parses from phrase structure parses. In *Proceedings of LREC*, volume 6, pages 449–454. Genoa Italy.

[12] Donahue, J., Jia, Y., Vinyals, O., Hoffman, J., Zhang, N., Tzeng, E., and Darrell, T. (2014). Decaf: A deep convolutional activation feature for generic visual recognition. In *International conference on machine learning*, pages 647–655.

[13] Gales, M. J. (1998). Maximum likelihood linear transformations for hmm-based speech recognition. *Computer speech & language*, 12(2):75–98.

[14] Gers, F. A. and Schmidhuber, E. (2001). Lstm recurrent networks learn simple context-free and context-sensitive languages. *IEEE Transactions on Neural Networks*, 12(6):1333–1340.

[15] Gers, F. A., Schmidhuber, J., and Cummins, F. (1999). Learning to forget: Continual prediction with lstm.

[16] Gers, F. A., Schraudolph, N. N., and Schmidhuber, J. (2002). Learning precise timing with lstm recurrent networks. *Journal of machine learning research*, 3(Aug):115–143.

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

[18] Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256.

[19] Goldberg, Y. and Levy, O. (2014). word2vec explained: deriving mikolov et al.'s negative-sampling word-embedding method. *arXiv preprint arXiv:1402.3722*.

[20] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

[21] Hecht-Nielsen, R. et al. (1988). Theory of the backpropagation neural network. *Neural Networks*, 1(Supplement-1):445–448.

[22] Hinton, G. E. (1984). Distributed representations.

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

[24] Huang, X., Acero, A., Hon, H.-W., and Foreword By-Reddy, R. (2001). *Spoken language processing: A guide to theory, algorithm, and system development*. Prentice hall PTR.

[25] Jurafsky, D. (2000). *Speech & language processing*. Pearson Education India.

[26] Karnaugh, M. (1953). The map method for synthesis of combinational logic circuits. *Transactions of the American Institute of Electrical Engineers, Part I: Communication and Electronics*, 72(5):593–599.

[27] Karpathy, A., Johnson, J., and Fei-Fei, L. (2015). Visualizing and understanding recurrent networks. *arXiv preprint arXiv:1506.02078*.

[28] Kingma, D. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[29] Koch, I. (2013). *Analysis of multivariate and high-dimensional data*, volume 32. Cambridge University Press.

[30] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

[31] Kruskal, J. B. and Wish, M. (1978). *Multidimensional scaling*, volume 11. Sage.

[32] Maaten, L. v. d. and Hinton, G. (2008). Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(Nov):2579–2605.

[33] Mikolov, T. (2012). Statistical language models based on neural networks. *Presentation at Google, Mountain View, 2nd April*.

[34] Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013a). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

[35] Mikolov, T., Karafiát, M., Burget, L., Cernockỳ, J., and Khudanpur, S. (2010). Recurrent neural network based language model. In *Interspeech*, volume 2, page 3.

[36] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013b). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

[37] Palangi, H., Deng, L., Shen, Y., Gao, J., He, X., Chen, J., Song, X., and Ward, R. (2016). Deep sentence embedding using long short-term memory networks: Analysis and application to information retrieval. *IEEE/ACM Transactions on Audio, Speech and Language Processing (TASLP)*, 24(4):694–707.

[38] Plaut, D. C. et al. (1986). Experiments on learning by back propagation.

[39] Ragni, A., Wu, C., Gales, M., Vasilakes, J., and Knill, K. (2017). Stimulated training for automatic speech recognition and keyword search in limited resource conditions. In *Acoustics, Speech and Signal Processing (ICASSP), 2017 IEEE International Conference on*, pages 4830–4834. IEEE.

[40] Sahidullah, M. and Saha, G. (2012). Design, analysis and experimental evaluation of block based transformation in mfcc computation for speaker recognition. *Speech Communication*, 54(4):543–565.

[41] Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958.

[42] Sundermeyer, M., Schlüter, R., and Ney, H. (2012). Lstm neural networks for language modeling. In *Thirteenth Annual Conference of the International Speech Communication Association*.

[43] Sutskever, I., Martens, J., Dahl, G., and Hinton, G. (2013). On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147.

[44] Tan, S., Sim, K. C., and Gales, M. (2015). Improving the interpretability of deep neural networks with stimulated learning. In *Automatic Speech Recognition and Understanding (ASRU), 2015 IEEE Workshop on*, pages 617–623. IEEE.

[45] Tang, Z., Shi, Y., Wang, D., Feng, Y., and Zhang, S. (2016). Visualization analysis for recurrent networks. Technical report, Tech. Rep., 2016, http://cslt. riit. tsinghua. edu. cn/mediawiki/images/6/6a/Visual. pdf.

[46] Tieleman, T. and Hinton, G. (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31.

[47] Toutanova, K., Klein, D., Manning, C. D., and Singer, Y. (2003). Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pages 173–180. Association for Computational Linguistics.

[48] Veselỳ, K., Ghoshal, A., Burget, L., and Povey, D. (2013). Sequence-discriminative training of deep neural networks. In *Interspeech*, pages 2345–2349.

[49] Werbos, P. J. (1990). Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560.

[50] Wu, C., Karanasou, P., Gales, M. J., and Sim, K. C. (2016). Stimulated deep neural network for speech recognition. In *INTERSPEECH*, pages 400–404.

[51] Xiong, W., Droppo, J., Huang, X., Seide, F., Seltzer, M., Stolcke, A., Yu, D., and Zweig, G. (2016). Achieving human parity in conversational speech recognition. *arXiv preprint arXiv:1610.05256*.

[52] Young, S., Evermann, G., Gales, M., Hain, T., Kershaw, D., Liu, X., Moore, G., Odell, J., Ollason, D., Povey, D., et al. (2002). The htk book. *Cambridge university engineering department*, 3:175.

[53] Zeiler, M. D. (2012). Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*.

[54] Zeiler, M. D. and Fergus, R. (2014). Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer.

[55] Zhang, C., Bengio, S., Hardt, M., Recht, B., and Vinyals, O. (2016). Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*.

Appendix A

POS Nomenclature

We followed the original POS Nomenclature of the Stanford POS parser, which is following reported for completeness. Only a subset of the original categories have been kept, which are evidenced in italics.

1. CC Coordinating conjunction
2. CD Cardinal number
3. DT Determiner
4. EX Existential there
5. FW Foreign word
6. IN Preposition or subordinating conjunction
7. JJ Adjective
8. JJR Adjective, comparative
9. JJS Adjective, superlative
10. LS List item marker
11. MD Modal
12. NN Noun, singular or mass
13. NNS Noun, plural
14. NNP Proper noun, singular
15. NNPS Proper noun, plural
16. PDT Predeterminer
17. POS Possessive ending
18. PRP Personal pronoun
19. PRP$ Possessive pronoun
20. RB Adverb
21. RBR Adverb, comparative
22. RBS Adverb, superlative
23. RP Particle

24. SYM Symbol
25. TO to
26. UH Interjection
27. *VB Verb, base form*
28. *VBD Verb, past tense*
29. *VBG Verb, gerund or present participle*
30. *VBN Verb, past participle*
31. *VBP Verb, non-3rd person singular present*
32. *VBZ Verb, 3rd person singular present*
33. *WDT Wh-determiner*
34. *WP Wh-pronoun*
35. *WP$ Possessive wh-pronoun*
36. *WRB Wh-adverb*