MAREK ZYLA

DEEP RECURRENT NEURAL NETWORKS IN CHARACTER-LEVEL LANGUAGE MODELING

MASTER THESIS

Kgs. Lyngby
2015, October

Marek Zyla
Deep Recurrent Neural Networks in character-level language modeling
Master thesis, 2015, October.

Technical University of Denmark
Department of Applied Mathematics and Computer Science
Richard Petersens Plads, building 324,2800 Kongens Lyngby, Denmark
Phone +45 45 25 30 31

WEBSITE:
http://www.compute.dtu.dk/

E-MAIL:
compute@compute.dtu.dk

ABSTRACT

Recurrent Neural Networks with gated units are now extremely popular due to their high performance on different tasks. This thesis attempts to analyze their behavior on the example of character-level language modeling. That is done by reproducing some of the latest findings, evaluating performance of different concepts and visualizing the internal mechanisms of the networks. It is presented how does the distribution of units in the layers influence performance and that making the networks deeper in different dimensions is beneficial. Analysis of the activations and weights exhibits that the models learn long term, interpretable dependencies and that the influence of the past sequences is the strongest in the top layers. Sampling experiment is conducted, which illustrates that the networks are able to spell correct English words and use punctuation. The neural models reveal their ability to preserve the context information and the units responsible for that phenomenon are identified. The dynamics of the training is visualized, which shows how the word dependencies affect their discovery time. It is shown that the representations created by the networks highly depend on the suffixes of the words.

ACKNOWLEDGEMENTS

I wish first of all to thank my supervisor, Ole Winther, for giving me the second chance and for offering the valuable feedback, especially at the end of the project. Also, I would like to thank my family for incessantly believing in me and my friends for providing me with influential discussions and delicious dinners.

CONTENTS

1 INTRODUCTION

1.1 MOTIVATION AND GOAL

Recurrent Neural Networks (RNNs) form a fairly novel approach in machine learning. They were first introduced in late 1980s (Schmidhuber, 2014). Since then, the architectures and algorithms have been far improved and computers have become millions of times more powerful. Only for the last few years it has been feasible to apply RNNs to real life problems.

Nowadays many researchers are achieving state-of-the-art results using different RNN architectures in solving various tasks: handwriting recognition and generation (Graves, 2013), learning word embeddings (Mikolov et al., 2013), machine translation (Bahdanau et al., 2014), speech recognition (Graves et al., 2013, Hannun et al., 2014), video analysis (Donahue et al., 2014), image captioning (Karpathy and Fei-Fei, 2014, Vinyals et al., 2014), protein secondary structure prediction (Kaae Sønderby and Winther, 2014), subcellular localization of proteins (Kaae Sønderby et al., 2015).

One conspicuous example of deep neural networks superiority is an architecture that in 2011 won the traffic sign recognition contest - IJCNN in San Jose. The goal of the contestants was to recognize traffic signs presented on images. The winning algorithm correctly classified 99.46% of the test images - more than the best individual human, who achieved 99.22% accuracy (Stallkamp et al., 2012). Figure 1 depicts a set of randomly chosen images from each sign category. The images were preprocessed - they were cut to contain the sign itself with a 10% margin around. It shows how limited the problem domain was. On the other hand, the blurriness and varying brightness of some instances show the difficulties the models had to cope with. The system was built from multiple stacked Convolutional Neural Networks (CNNs).

Even though the CNN differs from RNN in many aspects, they both share the basic principles. The provided example vividly presents the power of

Figure 1: Random representatives of the 43 traffic sign classes (from Stallkamp et al., 2012).

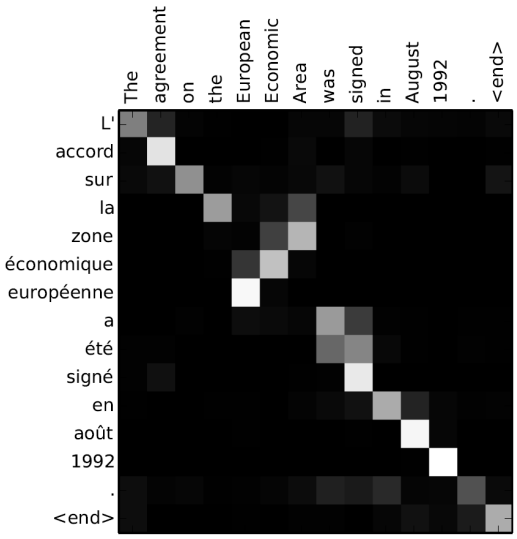


Figure 2: Alignment between an arbitrarily chosen English sentence and its French translation found by the RNN model (from Bahdanau et al., 2014).

Neural Networks in general. I believe RNNs still hold an unreleased potential. The second example is presented in order to justify that belief.

The most important advantage of RNNs, originating directly from their architecture, is their ability to deal with varying input length. Authors of Bahdanau et al., 2014 applied a special RNN architecture to English-to-French translation task. Their model not only learns to translate single words, but at the same time it tries to shift its focus on the corresponding parts of the sentence. Astonishing results were obtained. It was shown that the algorithm learns links that match human intuition.

Figure 2 depicts the alignment matrix between an arbitrarily chosen English sentence and its French translation. The brightness of the cell represents the correlation between particular word in the source sentence (English) and the generated translation (French). It can be seen that the number of words is not identical in both sentences. It is worth noticing that "European Economic Area" was correctly translated (and reorder) into "zone économique européenne". The most astonishing fact is that the network learned to translate English to French itself by simply looking at various pairs of English sentences and corresponding translations in French. Then it used its knowledge to translate previously unseen sentence (the sentence shown in the figure was not presented to the network during training).

However, we still lack specific knowledge on many aspects of how exactly Recurrent Neural Networks work. Hereby, it seems worthwhile to analyze the behavior of the RNNs. This thesis is an attempt to understand them more precisely and to get more insight into what affects their performance and how can we excel them. In order to achieve that goal some of the latest results are reproduced. Also various techniques and approaches are tested and evaluated. Some of them have already been tried on RNNs, others have not. From many available choices, language modeling task is used as the *only* test environment in this elaboration. Its relative computational simplicity and interpretability make it a perfect candidate.

1.2 STRUCTURE OF THE THESIS

The first chapter expresses author's motivation, presents two examples of Neural Networks application and identifies RNN analysis as the goal of the thesis .

Chapter 2 provides background information about machine learning, Neural Networks, computing power and presents the problem of language modeling that serves as a test framework.

Chapter 3 introduces the concepts used in the research and describes in detail the necessary theory: Recurrent Neural Networks, gated units, ambiguous depth and the training process.

The presented concepts are evaluated in chapter 4 with regards to their performance. The influence of the units distribution in layers and the models depth are examined.

Chapter 5 attempts to visualize the processes happening inside RNNs. The activations and weights are analyzed, sampling experiment is conducted, context memory, training dynamics and word representations are evaluated.

In chapter 6 a novel char-word architecture is proposed.

Chapter 7 provides a summary of the thesis and suggests areas for further study.

2 BACKGROUND

2.1 MACHINE LEARNING

The basic idea of machine learning is to tune parameters of a model (probably highly non-linear), so it implies desired behavior. If the goal is to map some input to target output it is referred to as *supervised* learning. During the training the model is exposed to corresponding input-output pairs, so it infers a general representation. Then, when previously unseen input data is presented, it is supposed to predict correct target. In case the input is supposed to be assigned to one of a finite set of categories it is a *classification* problem. If the output is a vector of continuous variables it is called a *regression* task (Bishop, 2006).

When only the input data is presented to the model, without any targets, the method is called *unsupervised* learning. It can be used to discover similarities in data (clustering), distribution of the data (density estimation) or dimensionality reduction (Bishop, 2006). Erhan et al., 2010 used unsupervised learning to pre-train some deep models before applying the supervised training.

Another method has been developed - *reinforcement* learning. The model is trying to find the best action it can take in a given situation in order to maximize a reward. It is not given any input-target data, but instead it has an ability to act in a predefined domain and collect the response. It has to learn its parameters in a trial and error process (Bishop, 2006).

The classification problem can be split into two phases. In the first one, the inference stage, the posterior probabilities $P(C_k|x)$ (probabilities of assigning to class C_k given the input x) are learned for each class k. In the second one, the decision stage, the probabilities are used to make an optimal decision. If the model learns not only the posterior probabilities, but also the distribution of the input, it is called generative. If only the posterior probabilities are learned, the model is referred to as discriminative. In case the model maps the inputs directly onto classes, without the use of probabilities, it is called a discriminative function (Bishop, 2006).

For the sake of this thesis, supervised learning will be used. As it is described later, language modeling is a classification task. Networks will represent a discriminative model and the decisions will be made by minimizing the misclassification rate, which is equivalent to choosing the class with highest posterior probability.

2.2 NEURAL NETWORKS

An artificial neural network (NN) is created by connecting many simple processing units (neurons) with each other. Neurons in the most exterior layer are fed with information from outside and based on it they produce activations. Each consecutive layer creates its own activations based on weighted activations from already activated neurons. The last layer produces output.

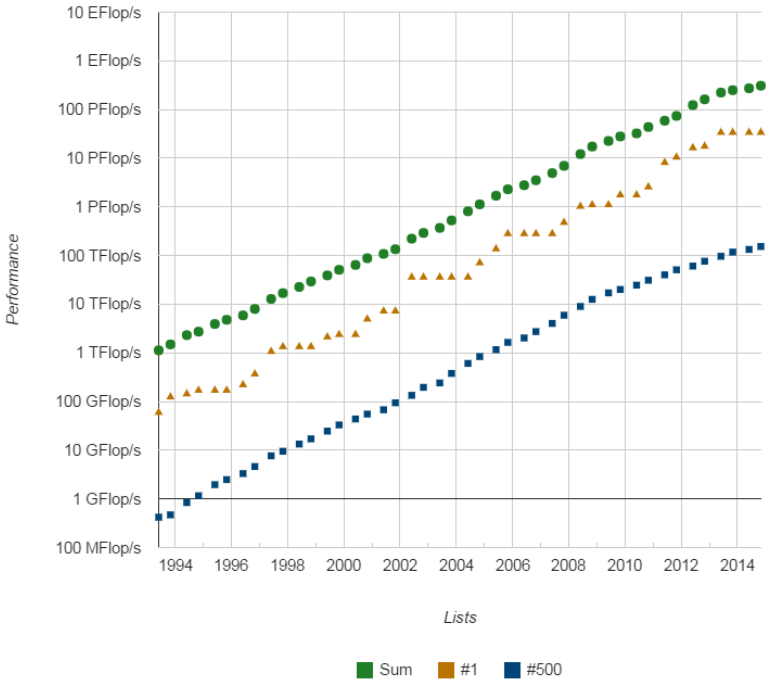


Figure 3: Exponential growth of supercomputing power as recorded by the TOP500 list (*Top 500 supercomputing sites*).

The simplest architecture is a feedforward neural network (FNN), where the connections between units do not form a directed cycle.

Training such networks is simply finding weights for all the connections that will effect in desired functioning. Depending on how exactly the neurons are connected and how they generate their activations (i.e. what nonlinearity they apply) it may be necessary to execute not one, but the whole chain of many causal computations in order to route activations from input to output. According to Schmidhuber, 2014 *deep* neural networks are ones, where there are *many* such steps required.

2.3 COMPUTING POWER

As it was stated before, only in the last few years the computers gained enough computational power to face demanding tasks as deep neural network training. Figure 3 presents how the supercomputing power has been increasing for last 2 decades. For comparison, PlayStation 4 has computing power of 1.84 TFlop/s (*PlayStation 4 Safety Guide* 2013), which is the equivalent of the best supercomputer on the planet in 1999 or the 500th best supercomputer in 2006. It is also worth noticing that the vertical axis is presented in logarithmic scale. The feasibility highly depends on the size of the problem and complexity of the task (number of network parameters), but with continuing growth of the computational power and decreasing prices, it is not only possible to train deep neural networks in general, but one can apply neural networks to solve some problems on a personal laptop computer.

2.4 LANGUAGE MODELING

As absurd as it may appear[1], language modeling is a task of finding the probability of a sequence of symbols (words or characters) $s_1 s_2 ... s_n$. This probability can be rewritten as a product of its components (Goodman, 2001):

$$P(s_1 s_2 ... s_t) = P(s_1) \cdot P(s_2|s_1) \cdot ... \cdot P(s_t|s_1 s_2 ... s_{t-1}) . \quad (1)$$

The model can be simplified to depend only on the $n-1$ previous symbols, a n-gram model:

$$P(s_t|s_1 s_2 ... s_{t-1}) = P(s_t|s_{t-(n-1)} ... s_{t-2} s_{t-1}) . \quad (2)$$

Usually, only the two previous symbols are used (trigram models). The trigram models are widely used due to their ratio of simplicity and performance. In this thesis more powerful models are used - Recurrent Neural Networks - that are trained using limited amount of previous characters (e.g. 100), but the trained models generalize and utilize information from theoretically unlimited number of symbols.

The obvious limitation of the n-gram models is lack of the context information. The benefits of such have been known for a long time now (Jelinek et al., 1991). On the other hand, there are disadvantages of using the context information as well. They can be presented on the example of speech recognition. In case some words have been incorrectly recognized, the probability of recognizing the next one correctly decreases, since the model is using invalid data from incorrectly picked up context.

Apart from speech recognition, language models are used in translation, human-machine interfaces (e.g. inputting text on a smartphone), handwriting recognition, information retrieval and more. The tasks are often split into two subtasks, from which one is to assign a probability of a sequence of symbols without any prior knowledge.

In this elaboration only character-level models are evaluated. Throughout the thesis, the set of all symbols appearing in a dataset will be called its vocabulary.

1 Noam Chomsky in 1969 said: "*But it must be recognized that the notion of 'probability of a sentence' is an entirely useless one, under any known interpretation of this term.*"

3 RECURRENT NEURAL NETWORKS

A recurrent neural network extends FNN by introducing units with a hidden state. It provides the ability to cope with variable-length sequences as input. Position in the sequence is referred to as time. In each time step the output and the hidden state of the unit are updated based on the previous hidden state and current position:

$$\begin{aligned} h_t &= f_1(h_{t-1}, x_t), \\ y_t &= f_2(h_t). \end{aligned} \tag{3}$$

f_1 and f_2 can be any differentiable nonlinear functions. In traditional RNNs f_1 is a sigmoid of weighted sum of the arguments and f_2 an identity function. RNN can be seen as a discrete-time dynamical system with an input x_t, an output y_t and a hidden state h_t (Pascanu et al., 2013).

Recurrent Neural Network can be visualized as in Graves, 2012, by unfolding the the network along the input sequence. It is presented in Figure 4.

Gradient descent methods are used to train Neural Networks. In 1980s a very efficient method, backpropagation (BP), gained popularity (Rumelhart et al., 1988). However, as it worked efficiently with FNNs, it was difficult to achieve sufficient results with RNNs. The reason for that problem was identified in Hochreiter, 1991 - the vanishing/exploding gradient while backpropagating in time or through multiple layers (Bengio et al., 1994).

There were many attempts to cope with the problem. Authors of Chung et al., 2014 divide them into two main categories. First is to develop a more efficient learning algorithm (Bengio et al., 2012). The second proposes usage of a different unit architecture, with incorporated gating mechanism that allows to control the flow of the activations in the network based on the previous time-steps. These gated units are introduced in the following section.

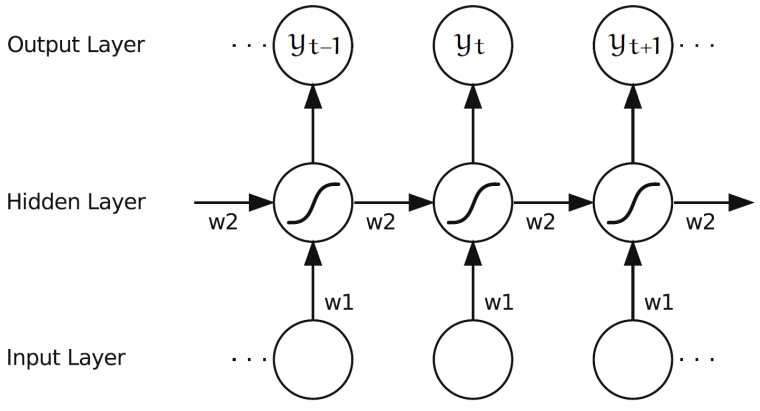


Figure 4: Unfolded traditional Recurrent Neural Network. Each node represents state of the layer at given time step. Weights from input to hidden layer are denoted as w1, recurrent connections (from previous hidden state to current one) are labelled w2. The same weights are used at each time step. Adapted from Graves, 2012 to match nomenclature.

3.1 GATED UNITS

3.1.1 LSTM

One of the gated units is Long short-term memory unit (LSTM), introduced in Hochreiter and Schmidhuber, 1997. A LSTM unit is equipped with a memory cell and gating mechanism that enables it to hold or forget the information stored. Therefore its state is not overwritten in every time-step like in standard RNN, but it can be remembered throughout the computation. It is possible to store this information for long periods of time. It becomes then a differentiable equivalent of a computer memory cell (Graves, 2012). Researchers developed and evaluated many variants of the unit, but the basic idea remains the same. A thorough analysis of different variants was conducted in Greff et al., 2015.

For the sake of this thesis the LSTM block is defined as in Graves, 2012, but assuming only one cell per unit. The equations are given for a single LSTM block. w_{ij} is weight of the connection from unit i to j. a_j^t is input to the unit j at time t and b_j^t is activation of unit j at time t. The subscripts ι, ϕ, ω and c are respectively the input gate, forget gate, output gate and memory cell. The peephole weights from the cell to the gates are denoted $w_{c\iota}$, $w_{c\phi}$ and $w_{c\omega}$. s_c^t refers to the state of the cell at time t. f, g, h refer to the activation function of the gates, cell input activation and output activation function, respectively. I is the number of inputs and H - number of hidden units.

Input gates

$$a_\iota^t = \sum_{i=1}^{I} w_{i\iota} x_i^t + \sum_{h=1}^{H} w_{h\iota} b_h^{t-1} + w_{c\iota} s_c^{t-1}\,, \tag{4}$$

$$b_\iota^t = f(a_\iota^t)\,. \tag{5}$$

Forget gates

$$a_\phi^t = \sum_{i=1}^{I} w_{i\phi} x_i^t + \sum_{h=1}^{H} w_{h\phi} b_h^{t-1} + w_{c\phi} s_c^{t-1}\,, \tag{6}$$

$$b_\phi^t = f(a_\phi^t)\,. \tag{7}$$

Cells

$$a_c^t = \sum_{i=1}^{I} w_{ic} x_i^t + \sum_{h=1}^{H} w_{hc} b_h^{t-1}\,, \tag{8}$$

$$s_c^t = b_\phi^t s_c^{t-1} + b_\iota^t g(a_c^t)\,. \tag{9}$$

Output gates

$$a_\omega^t = \sum_{i=1}^{I} w_{i\omega} x_i^t + \sum_{h=1}^{H} w_{h\omega} b_h^{t-1} + w_{c\omega} s_c^t , \tag{10}$$

$$b_\omega^t = f(a_\omega^t) . \tag{11}$$

Cell outputs

$$b_c^t = b_\omega^t h(s_c^t) . \tag{12}$$

Figure 5 depicts the schematic of a LSTM block. There are three gates controlling information flow: input, output and forget gate. Output of each gate is bounded by 0 (blocking the flow) and 1 (uninterrupted flow). The input to the block at time t is either the input from data itself or the output from the blocks in the previous layer in time t. In each time step, the content of the internal memory cell is updated with the new input. The impact of the input is controlled with the input gate. The factor, with which the previous state is being forgotten is controlled by the forget gate. Usually the output activation function is a hyperbolic tangent, so the output from the block is scaled to be between -1 and 1. Also, this is the case with the change of the cell memory content in one time step, because of the input activation function. The gating mechanism allows LSTM blocks to store or integrate information.

Some researchers (Zaremba and Sutskever, 2014, Sutskever et al., 2014, Kaae Sønderby and Winther, 2014) do not use the peephole connections,

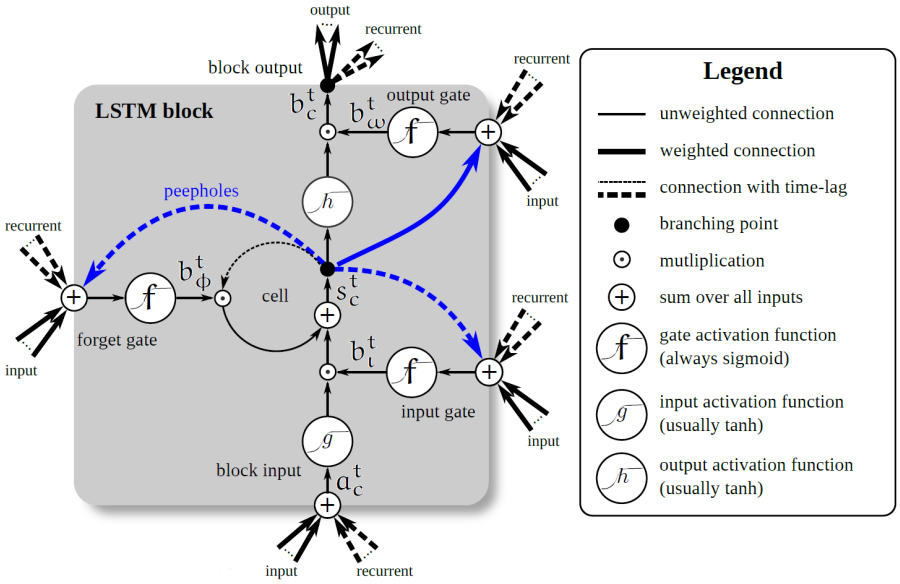


Figure 5: Schematic of a LSTM block. Sum over all inputs includes bias parameters (not presented in the figure). Adapted from Greff et al., 2015 to match nomenclature.

as on some datasets they do not provide higher performance. Authors of Greff et al., 2015 tested how the peepholes affect the results on three datasets. They report 2 cases of peepholes improving performance and 1 case causing otherwise. The differences between performance in the 3 cases were small. The authors also conclude that the forget gate and the output activation function are the most critical components of the LSTM block.

3.1.2 GRU

Recently, another solution, Gated Recurrent Unit (GRU), was proposed in Cho et al., 2014. The architecture was motivated by LSTM, but it is simpler in implementation and computation - it contains only 2 gates - update and reset gate.

In this thesis the GRU unit is defined as in Chung et al., 2014 with candidate activation as in Equation 15 (there are two very similar definitions in the paper). The equations are presented for a single GRU unit. Unit may refer to the whole GRU block or its part that produces activation. w_{ij} is weight of the connection from unit i to j. a_j^t represents input to the unit j at time t and b_j^t is an activation of the unit j at time t. The subscripts r, κ, u describe respectively the reset gate, candidate activation and update gate. The activation functions of the gates and the candidate are denoted respectively as f and g. I is the number of inputs and H - number of hidden units in the layer.

Reset gates

$$a_r^t = \sum_{i=1}^{I} w_{ir} x_i^t + \sum_{h=1}^{H} w_{hr} b_h^{t-1}, \tag{13}$$

$$b_r^t = f(a_r^t). \tag{14}$$

Candidate activations

$$a_\kappa^t = \sum_{i=1}^{I} w_{i\kappa} x_i^t + b_r^t \sum_{h=1}^{H} w_{h\kappa} b_h^{t-1}, \tag{15}$$

$$b_\kappa^t = g(a_\kappa^t). \tag{16}$$

Update gates

$$a_u^t = \sum_{i=1}^{I} w_{iu} x_i^t + \sum_{h=1}^{H} w_{hu} b_h^{t-1}, \tag{17}$$

$$b_u^t = f(a_u^t). \tag{18}$$

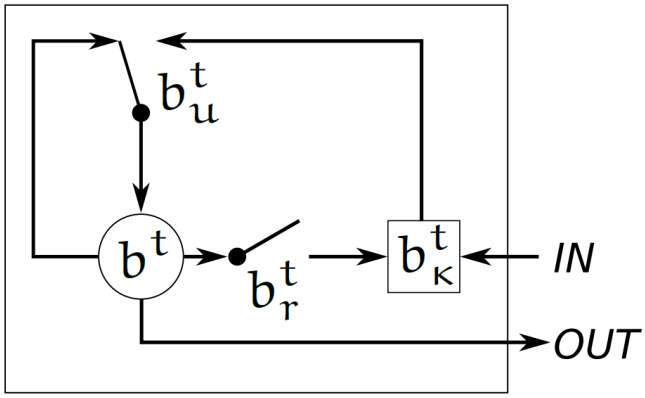


Figure 6: Schematic of a GRU block. Adapted from Chung et al., 2014 to match nomenclature.

Unit activations

$$b^t = (1 - b_u^t)b^{t-1} + b_u^t b_\kappa^t . \tag{19}$$

Figure 6 presents the schematic of a GRU unit. The reset gate selects whether to ignore the previous state or to use it in calculation of a new hidden state candidate. The update gate controls to what extent the hidden state should be overwritten with the candidate - new activation is a linear interpolation of the candidate and activation in the previous time step. The output from the unit is the hidden state itself.

3.1.3 LSTM vs. GRU

Both LSTM and GRU improve performance in comparison to traditional RNN. However, superiority of one gating unit over another is inconclusive. Evaluation on the task of sequence modeling on datasets of raw speech signal and polyphonic music data was conducted in Chung et al., 2014. Authors suggest that the choice of the architecture may depend on the task and data.

3.1.4 Bidirectional LSTM/GRU networks

Access to both past and future context may be important in some tasks, hence bidirectional RNN was introduced in Schuster and Paliwal, 1997. The idea was evaluated with LSTM in Graves and Schmidhuber, 2005. The authors report performance increase on a task of phoneme classification compared to unidirectional networks. GRU networks should benefit similarly. It is a powerful and useful concept, but in case of language modeling it usually cannot be applied, since language models are used for prediction, rather than filling a gap. If there is more than one symbol in the gap, as stated in Berglund et al., 2015, it becomes an even less trivial task and special frameworks are necessary. In this thesis only unidirectional networks are used.

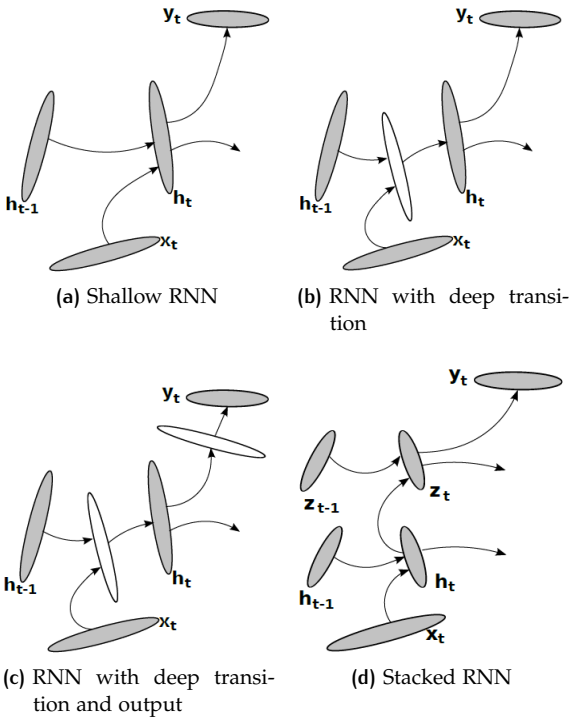
3.2 THE MEANING OF DEPTH

As pointed out in Pascanu et al., 2013, the meaning of depth in the case of RNNs is ambiguous. They examined and empirically tested the effects of making a RNN deeper in different orthogonal dimensions. The first approach was to insert a feedforward network with one or more hidden layers between hidden state and output (*deep output*) or between consecutive hidden states (*deep transition*). The next one was to build a network with layers stacked on each other (*stacked RNN*). Each augmentation was supposed to bring its own value to the model and thereby increase the performance. All approaches are presented in Figure 7.

The models were tested on two datasets using traditional RNNs (no gating mechanism). In polyphonic music prediction task the augmentations brought either only low performance increase or even a decrease (in 25% of the cases). Only the use of another architecture - a network augmented in all 3 dimensions, with special nonlinearities (Gulcehre et al., 2013, Goodfellow et al., 2013) and the use of dropout generalization technique (Hinton et al., 2012) - delivered a significantly higher performance. As an outcome, the results are not easily comparable, since the special units and dropout were used only in the one test case. It cannot be concluded how each augmentation affects the network, but in general the deeper the network, the higher the performance on complex tasks.

In language modeling task the performance increase gained by such augmentations is more straightforward. On word-level model authors were able to outperform shallow (1 layer) LSTM structure and deliver state-of-the-art results.

It would be worthwhile to examine the effects of making LSTM/GRU networks deeper in such manners. Even though some researchers use dif-



(a) Shallow RNN

(b) RNN with deep transition

(c) RNN with deep transition and output

(d) Stacked RNN

Figure 7: Various meanings of depth applied to RNNs (from Pascanu et al., 2013).

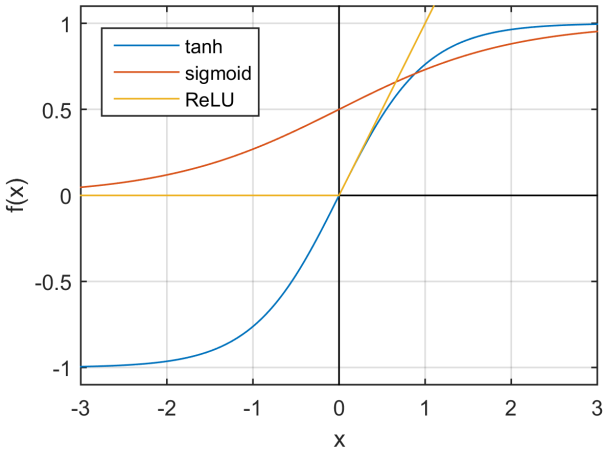


Figure 8: Two standard nonlinearities used in RNNs (logistic sigmoid, hyperbolic tangent) and Rectifier Linear Unit (ReLU).

ferent variants of deep LSTM/GRU networks (Kaae Sønderby and Winther, 2014, Karpathy et al., 2015), to the author's knowledge, there is no thorough comparison yet.

3.3 NONLINEARITIES

The standard nonlinearities used in RNNs, also in LSTM/GRU units, are logistic sigmoid and hyperbolic tangent. They are both presented in Figure 8. In LSTM/GRU architectures logistic sigmoids control the flow of signals by opening (1) or closing (0) the gates in the extrema. On the other hand hyperbolic tangent scales the whole real domain to a value between -1 and 1. It is usually used as the output activation in LSTM or candidate activation in GRU. A few extra activation functions have been developed and they are briefly described below.

3.3.1 Rectifier Linear Unit

Rectifier Linear Unit (ReLU) was introduced in Glorot et al., 2011. It was motivated by a model of a human neuron. Like in a human brain, it brings sparsity in activations. The function is presented in Figure 8. It was evaluated on stacked auto-encoders (architecture very close to Feed Forward Network). Authors conclude that ReLU units could be beneficial in text mining tasks.

The nonlinearity increased performance in some scenarios, but to my knowledge, it has not been evaluated thoroughly with RNNs yet. It could be worthwhile to compare its performance and influence on RNNs. Since it is not bounded from the top, it should be applied in deep transition or deep output function.

3.3.2 Output layer

Since the classification task is being researched in this work, there is another nonlinearity - a soft-max layer - placed (stacked) on the top of the network. For a multiclass classification problem, in which 1 of K classes is predicted

for each input symbol and the classes are mutually exclusive, each of K outputs is defined as

$$y_k(x, W) = \frac{exp(a_k(x, W))}{\sum_{j=1}^{K} exp(a_j(x, W))}, \tag{20}$$

where a_j are the activations from the layer, x - input and W - the weights of the network. Worth mentioning are the facts that $0 \leqslant y_k \leqslant 1$ and $\sum_k y_k = 1$.

3.4 TRAINING

Training Recurrent Neural Network is a task of finding its weights that will minimize some cost function. It may be done using gradient descent methods. The basic idea behind such methods is to find derivatives of the cost function (see Subsection 3.4.2) with regards to the network weights and then update the weights in the negative direction of the gradient.

3.4.1 Experimental setup

Datasets are always divided into 3 subsets: training, test and validation. Train set is used to train the model parameters. Validation set is used to tune hyperparameters, i.a. learning rate, dropout fraction. The test set serves for reporting purposes - for the best performance on validation set the result on the test set is reported (generalization performance). Usually, the validation and test sets are 5% or 10% of all data available each, and the rest is used for training.

Since the amount of the training data can be enormous (it would be impossible to fit some datasets in memory and train it at once), the training happens in batches. It also makes the learning faster. In each iteration of training only a portion of data is used to compute gradients and updates. Completing iterations through the whole dataset is called an epoch. Usually, it takes up to few hundreds epochs to converge. However, already below 100 epochs the results can be very close to convergence. Also, for limited memory reason, the backpropagation in time is also truncated, e.g. to 100 time steps.

Validation set can be used for early stopping. During the training, if the validation set cost stops to get better for a fixed amount of epochs (e.g. 10) the training is stopped.

When the training starts, the network parameters (weights and biases) are initialized to some value. Although there were many different papers published on this topic, in this thesis solution used in Karpathy et al., 2015 will be applied, which is to simply set all the weights with uniform distribution in range [-0.8, 0.8].

3.4.2 Cost function

In this elaboration the cross entropy is used as the cost function. One of the advantages of using cross entropy is that it is monotonic, in some sense, with a distance from the "true" model of the data (Goodman, 2001), with

distance meaning how many bits extra are needed to code a given output compared to the "real" model of the data. It is defined as

$$CE = -\sum_{t=1}^{N} \ln P(s_t|s_1 s_2 ... s_{t-1}) . \tag{21}$$

In this equation only probabilities assigning a correct label are used. The same function is used for evaluation of the models.

3.4.3 Backpropagation through time

To efficiently calculate the gradient, backpropagation through time (BPTT) is used. Author of Graves, 2012 points out that this method requires only $O(N)$ time compared to $O(N^2)$ when calculating the gradient in a straightforward manner. BPTT is basically the chain rule

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x} \tag{22}$$

applied repeatedly to each unit in each time step, starting from the last time step T and decrementing t. Let us present it on the example of a single layer LSTM network.

First let us define

$$\epsilon_c^t \stackrel{\text{def}}{=\!=} \frac{\partial \mathcal{L}}{\partial b_c^t} , \tag{23}$$

$$\epsilon_s^t \stackrel{\text{def}}{=\!=} \frac{\partial \mathcal{L}}{\partial s_c^t} , \tag{24}$$

$$\delta_j^t \stackrel{\text{def}}{=\!=} \frac{\partial \mathcal{L}}{\partial a_j^t} , \tag{25}$$

where $\mathcal{L}$ is the cost function. Based on the δ_j^t terms it will be possible to calculate derivatives with respect to the network weights. K represents the number of units outputs. The partial derivative of the loss function with regards to cell output depends on its influence on the output layer and on the hidden layer in the next time step:

$$\epsilon_c^t = \frac{\partial \mathcal{L}}{\partial b_c^t} = \sum_{k=1}^{K} \frac{\partial a_k^t}{\partial b_c^t}\frac{\partial \mathcal{L}}{\partial a_k^t} + \sum_{h=1}^{H} \frac{\partial a_h^{t+1}}{\partial b_c^t}\frac{\partial \mathcal{L}}{\partial a_h^{t+1}} = \\ = \sum_{k=1}^{K} w_{ck}\delta_k^t + \sum_{h=1}^{H} w_{ch}\delta_h^{t+1} . \tag{26}$$

Applying the chain rule to the δ term for the output gate input in time t and substituting Equation 12 and Equation 11 we get

$$\delta_\omega^t = \frac{\partial \mathcal{L}}{\partial a_\omega^t} = \frac{\partial b_c^t}{\partial a_\omega^t}\frac{\partial \mathcal{L}}{\partial b_c^t} = \frac{\partial}{\partial a_\omega^t}\left(b_\omega^t h(s_c^t)\right)\frac{\partial \mathcal{L}}{\partial b_c^t} = \\ = \frac{\partial}{\partial a_\omega^t}\left(f(a_\omega^t)h(s_c^t)\right)\frac{\partial \mathcal{L}}{\partial b_c^t} = f'(a_\omega^t)h(s_c^t)\epsilon_c^t . \tag{27}$$

The second ϵ term (with regards to cell state) depends on the influence from the cell output, cell state in the next time step, input gate in the next time step, forget gate in the next time step and output gate in the current time step:

$$\epsilon_s^t = \frac{\partial \mathcal{L}}{\partial s_c^t} = \frac{\partial b_c^t}{\partial s_c^t}\frac{\partial \mathcal{L}}{\partial b_c^t} + \frac{\partial s_c^{t+1}}{\partial s_c^t}\frac{\partial \mathcal{L}}{\partial s_c^{t+1}} + \frac{\partial a_\iota^{t+1}}{\partial s_c^t}\frac{\partial \mathcal{L}}{\partial a_\iota^{t+1}} + \\ + \frac{\partial a_\phi^{t+1}}{\partial s_c^t}\frac{\partial \mathcal{L}}{\partial a_\phi^{t+1}} + \frac{\partial a_\omega^t}{\partial s_c^t}\frac{\partial \mathcal{L}}{\partial a_\omega^t}. \quad (28)$$

Substituting Equation 12 to the partial derivative of cell output in time t with respect to cell state in time t gives

$$\frac{\partial b_c^t}{\partial s_c^t} = \frac{\partial}{\partial s_c^t}\left(b_\omega^t h(s_c^t)\right) = h'(s_c^t) b_\omega^t . \quad (29)$$

Substituting Equation 9 to the partial derivative of cell state in time $t+1$ with respect to cell state in time t gives

$$\frac{\partial s_c^{t+1}}{\partial s_c^t} = \frac{\partial}{\partial s_c^t}\left(b_\phi^{t+1} s_c^t + b_\iota^{t+1} g(a_c^{t+1})\right) = b_\phi^{t+1} . \quad (30)$$

Substituting Equation 4 to the partial derivative of input gate input in time $t+1$ with respect to cell state in time t gives

$$\frac{\partial a_\iota^{t+1}}{\partial s_c^t} = \frac{\partial}{\partial s_c^t}\Big(\sum_{i=1}^{I} w_{i\iota} x_i^{t+1} + \sum_{h=1}^{H} w_{h\iota} b_h^t + w_{c\iota} s_c^t\Big) = w_{c\iota} . \quad (31)$$

Substituting Equation 6 to the partial derivative of forget gate input in time $t+1$ with respect to cell state in time t gives

$$\frac{\partial a_\phi^{t+1}}{\partial s_c^t} = \frac{\partial}{\partial s_c^t}\Big(\sum_{i=1}^{I} w_{i\phi} x_i^{t+1} + \sum_{h=1}^{H} w_{h\phi} b_h^t + w_{c\phi} s_c^t\Big) = w_{c\phi} . \quad (32)$$

Substituting Equation 10 to the partial derivative of output gate input in time t with respect to cell state in time t gives

$$\frac{\partial a_\omega^t}{\partial s_c^t} = \frac{\partial}{\partial s_c^t}\Big(\sum_{i=1}^{I} w_{i\omega} x_i^t + \sum_{h=1}^{H} w_{h\omega} b_h^{t-1} + w_{c\omega} s_c^t\Big) = w_{c\omega} . \quad (33)$$

Substituting Equations 29, 30, 31, 32 and 33 into Equation 28:

$$\epsilon_s^t = h'(s_c^t) b_\omega^t \epsilon_c^t + b_\phi^{t+1} \epsilon_s^{t+1} + w_{c\iota}\delta_\iota^{t+1} + w_{c\phi}\delta_\phi^{t+1} + w_{c\omega}\delta_\omega^t . \quad (34)$$

Applying the chain rule to the δ term for the cell input in time t and substituting Equation 9:

$$\delta_c^t = \frac{\partial \mathcal{L}}{\partial a_c^t} = \frac{\partial s_c^t}{\partial a_c^t}\frac{\partial \mathcal{L}}{\partial s_c^t} = \frac{\partial}{\partial a_c^t}\left(b_\phi^t s_c^{t-1} + b_\iota^t g(a_c^t)\right)\frac{\partial \mathcal{L}}{\partial s_c^t} = b_\iota^t g'(a_c^t)\epsilon_s^t . \quad (35)$$

Applying the chain rule to the δ term for the forget gate input in time t and substituting Equation 9:

$$\delta_\phi^t = \frac{\partial \mathcal{L}}{\partial a_\phi^t} = \frac{\partial s_c^t}{\partial a_\phi^t}\frac{\partial \mathcal{L}}{\partial s_c^t} = \frac{\partial}{\partial a_\phi^t}\left(b_\phi^t s_c^{t-1} + b_\iota^t g(a_c^t)\right)\frac{\partial \mathcal{L}}{\partial s_c^t} = \\ = \frac{\partial}{\partial a_\phi^t}\left(f(a_\phi^t)s_c^{t-1} + b_\iota^t g(a_c^t)\right)\frac{\partial \mathcal{L}}{\partial s_c^t} = f'(a_\phi^t)s_c^{t-1}\epsilon_s^t\,. \quad (36)$$

Applying the chain rule to the δ term for the input gate input in time t and substituting Equation 9 gives

$$\delta_\iota^t = \frac{\partial \mathcal{L}}{\partial a_\iota^t} = \frac{\partial s_c^t}{\partial a_\iota^t}\frac{\partial \mathcal{L}}{\partial s_c^t} = \frac{\partial}{\partial a_\iota^t}\left(b_\phi^t s_c^{t-1} + b_\iota^t g(a_c^t)\right)\frac{\partial \mathcal{L}}{\partial s_c^t} = \\ = \frac{\partial}{\partial a_\iota^t}\left(b_\phi^t s_c^{t-1} + f(a_\iota^t)g(a_c^t)\right)\frac{\partial \mathcal{L}}{\partial s_c^t} = f'(a_\iota^t)g(a_c^t)\epsilon_s^t\,. \quad (37)$$

Since all the δ terms are now available they can be used to derive partial derivatives with regards to the network weights:

$$\frac{\partial \mathcal{L}}{\partial w_{ij}} = \sum_{t=1}^{T}\frac{\partial \mathcal{L}}{\partial a_j^t}\frac{\partial a_j^t}{\partial w_{ij}}\,, \quad (38)$$

where

$$\forall_j : \delta_j^{T+1} = 0 \quad (39)$$

and all states and activations are initialized to 0 at $t = 0$. These derivatives are used to calculate the updates.

3.4.4 Updates

There are many different algorithms that based on the computed gradients deliver new values for trainable parameters. Only one is used in this thesis - RMSProp.

RMSProp was introduced in Tieleman and Hinton, 2012 and it has been proven to be an efficient method for batch training. The basic idea is to calculate the updates separately for each weight. It turns out that, instead of making them proportional to the gradient, it is more efficient to use the sign of the gradient and some step size. It is equivalent to using the gradient and then dividing by the size of the gradient. In order to achieve it in batch training, a moving average of squared errors is kept for each weight w:

$$MS(w, b) = \rho MS(w, b-1) + (1-\rho)\left(\frac{\partial \mathcal{L}}{\partial w}(b)\right)^2\,, \quad (40)$$

where b denotes the batch number, ρ is a parameter and $\mathcal{L}$ - the cost function.

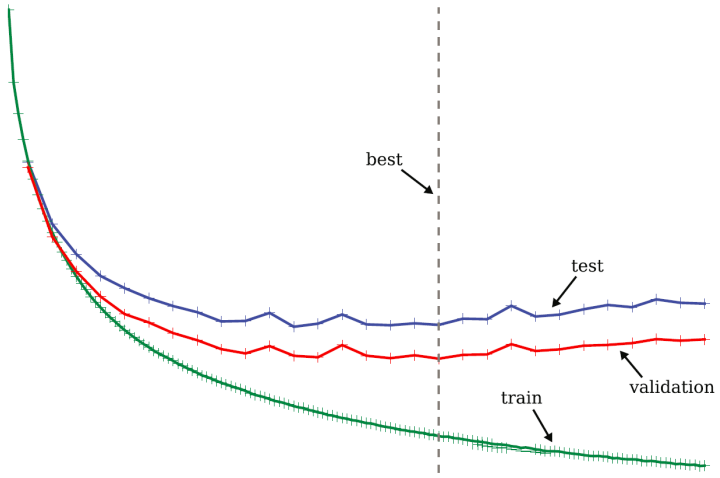


Figure 9: An example of overfitting when training neural networks (from Graves, 2012).

Root square of the expression above approximates the average gradient for weight w over the whole training set. Therefore, the updates are calculated as:

$$u(w, b) = \sigma \frac{\frac{\partial \mathcal{L}}{\partial w}(b)}{\sqrt{MS}}, \tag{41}$$

where σ is a learning rate parameter.

3.4.5 Inductive bias

According to Mitchell, 1997 an *inductive bias* of a learning algorithm is a set of additional assumptions that are sufficient to infer output for unseen data.

During learning, the algorithm is presented some data. Then, the model is supposed to predict the correct output also for the data that it has not seen before. It is only possible if some assumptions are made. These assumptions are called the inductive bias.

One example of the inductive bias is Occam's Razor, which says that the simplest (valid for already seen data) solution is the best.

In case of RNNs, the inductive bias is probably yielded by the structure of the model and the cost function. The stacked networks can extract some features on lower layers that are reused in various configurations by the units in the higher layers. It is likely that seen and unseen data share some of such features, even if in different combinations. The hierarchical structure allows the model to exploit these features to predict output for previously unseen data. Also, since in this thesis the cross entropy is used as a cost function, the inductive bias should promote solutions, in which the outputs can in average be coded on smaller number of bits. It is equivalent to encouraging higher compression.

3.5 GENERALIZATION TECHNIQUES

Training large, deep neural networks on relatively small datasets make them tend to perform rather poorly on previously unseen data. This phenomenon - *overfitting* - is one of the main problems connected with neural networks. Figure 9 depicts an exemplary correct classification rate (CCR) plot as a function of number of training epochs. It can be seen that at some point performance on the validation set starts to drop, even though the train set CCR keeps improving. It is a typical situation, where overfitting takes place. To overcome this obstacle, many generalisation techniques have been proposed. The most straightforward one is to bring more training data (Bishop, 2006), but that is not always feasible. One of the most popular methods, that improves generalization independently of increasing the size of the dataset, is dropout.

3.5.1 Dropout

This method was introduced in Hinton et al., 2012 for FNN. Now it has been accepted as a standard generalization technique that is often applied in the neural network research.

Dropout randomly omits a part of the activations in the neural network in each training batch by setting them to zero. It can be seen as choosing only a subset of hidden units and, therefore, choosing one model from a set of different neural models that share the same weights. It encourages every single hidden unit to learn a useful activation (feature extraction) that does not rely on other hidden units and that is correct in many different contexts.

Since its introduction, dropout did not perform well with RNNs. Authors of Zaremba et al., 2014 adapted it to improve generalization also in this architecture. The solution is to apply dropout only to non-recurrent connections of the network. The idea is presented in Figure 10.

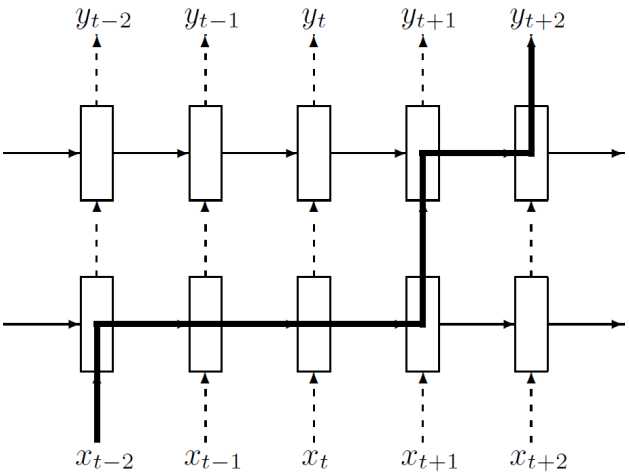


Figure 10: Dropout adapted to Recurrent Neural Network. The bold line presents exemplary information flow through the network. Thin solid line depicts a recurrent connection. The dropout is applied only to non-recurrent connections (dashed line). From Zaremba et al., 2014.

4 PERFORMANCE EVALUATION

Some experiments were conducted in an attempt to analyze RNN performance. The models were examined on one dataset - a novel "War and Peace" by Leo Tolstoy.

The input to the network is a sequence of characters and the goal is to predict the next character in the sequence at each time step, as presented in Figure 11. The input is encoded into a sequence of 1-hot K-dimensional vectors, each representing one character, where K is the size of the vocabulary. This sequence is fed to the network. Softmax classifier on the top of the network for each time step produces a vector of probabilities of assigning each character as the next one in the sequence. The objective of the network is to minimize the average cross entropy over the whole dataset.

All parameters are initialized by sampling from uniform distribution in range [-0.08, 0.08]. Batch of size 100 is used during training. The gradient is propagated for 100 time steps and clipped elementwise in range [-5, 5]. RMSProp algorithm is used to update the network parameters based on the gradients. Base learning rate 0.002 is used. The network is trained for 50 epochs and after training for 10 epochs the learning rate is multiplied by 0.95 after each additional epoch. All parameters are taken from Karpathy et al., 2015. The dropout fraction of 8% is used for each model. Hidden state in the first batch is initialized to zero. Each next batch is initialized with the last hidden state of the previous batch. Therefore, even though the gradient is backpropagated to 100 steps, the hidden states have values as if the network was reading up to the last 26000 characters (there is 2.6 million characters in the whole training set and batches contain 100 lines each). No peephole connections were used. These parameters are used in all the experiments, unless stated otherwise.

The names of the examined networks follow a pattern *NarchitectureS*, where N is the number of layers, architecture is one of LSTM, GRU, RNN and S represents the number of hidden units. In order to easily compare the results with those from Karpathy et al., 2015 the same notation of S is used. The number of hidden units is chosen, so that the number of parameters is equivalent to number of parameters in 1 layer LSTM network with that number of units on "War and Peace" dataset. The dataset influences the number of the units due to different vocabulary size (87 characters in "War and Peace"). The resulting numbers of parameters were approximately 50K, 130K, 400K and 1.3M for 64, 128, 256 and 512 units, respectively.

4.1 IMPLEMENTATION

All experiments in this and following chapters were implemented in Python using Lasagne (Dieleman et al., 2015). Lasagne is a lightweight library to build Neural Networks in Python. It is still in development, version 0.2dev1 was used in this thesis. It is built upon Theano, another library that allows symbolic mathematical computation. The main benefit of Theano utilized

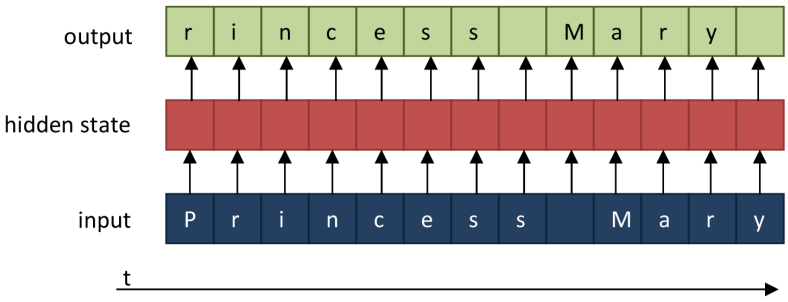


Figure 11: Schematic of the architecture.

in Lasagne is symbolic differentiation and feasible GPU usage. It allowed RNN training on a personal laptop with dedicated Nvidia graphic card.

Lasagne implements both feedforward and recurrent layers, i.a. LSTM, GRU. Since Lasagne is still in development, some functionality has not been implemented yet. Deep hidden transition is one of them and it had to be implemented alongside the thesis (Section 4.4). All necessary helper functions like sampling and visualization were written together with the thesis. The plots were mainly produced using MATLAB R2015a, but also matplotlib library and HTML.

Once one understands Theano and Lasagne, the implementation of the experiments is fairly straightforward. Furthermore, the implementation is not the goal of the thesis, rather it was created as a tool. Therefore, information provided about it is very limited. The code used for training the 3LSTM64 is presented in Appendix B. The file was built upon *pentree.py* example (Kaae Sønderby, 2015).

4.2 REPRODUCTION OF KARPATHY'S RESULTS

LSTM and GRU models with sizes 64, 128, 256 and 512 were trained, as in Karpathy et al., 2015. The results are presented in Table 6. The original Karpathy's results are presented in Table 2. Kartpathy's vanilla RNN results are included for comparison. The exact network sizes are presented in Appendix D.

Majority of the reproduced results is worse than Karpathy's, only 8.33% of the models delivered lower cross entropy (1LSTM512 and 2GRU512). The lowest cross entropy was achieved by the 3LSTM512 model with value 1.100, compared to 1.077 using 3GRU512 in Karpathy's.

The differences in the cross entropy values are most likely due to different dropout fractions used. Karpathy used early stopping to cross-validate the

Table 1: Test set cross entropy for different models (reproduced results).

	LSTM			GRU		
Layers	1	2	3	1	2	3
Size						
64	1.496	1.485	1.570	1.484	**1.460**	1.525
128	1.329	**1.284**	1.370	1.304	1.306	1.324
256	1.196	**1.178**	1.210	1.204	1.186	1.214
512	1.153	1.128	**1.110**	1.181	1.145	1.134

Table 2: Test set cross entropy for different models (Karpathy's results).

	LSTM			RNN			GRU		
Layers	1	2	3	1	2	3	1	2	3
Size									
64	1.449	1.442	1.540	1.446	1.401	1.396	1.398	**1.373**	1.472
128	1.277	**1.227**	1.279	1.417	1.286	1.277	1.230	**1.226**	1.253
256	1.189	**1.137**	1.141	1.342	1.256	1.239	1.198	1.164	**1.138**
512	1.161	1.092	1.082	-	-	-	1.170	1.201	**1.077**

models and find best dropout values separately for each model. However, they are not reported in the paper. Due to limited amount of time only few values of dropout were validated and only for a single network. Then the same dropout fraction (8%) was used to train all models. However, it clearly can be seen that the values are comparable and scale similarly.

It is easy to notice from Table 6 that the factor that affects the performance the most is the size of the network. Except one case (2LSTM256 outperforms 1GRU512) all models with more parameters delivered better results than smaller models in any configuration.

The results delivered using LSTM and GRU units are fairly similar. A trend can be spotted that smaller models perform better using GRUs and bigger ones with LSTMs.

It can be seen that the models with more than 1 layer outperform the single layered networks (with exception of GRU128). It cannot be stated if 2 or 3 layers are better. The largest models obtained higher performance using 3 layers and all smaller networks brought lower cross entropy with 2 layers. These findings are similar to Karpathy's.

The hidden states in the first batch in cross entropy evaluation are always initialized with zeros. The network does not have information about the past characters, so it delivers lower performance. Therefore, it was checked how does it influence the results to use a burnout of 2 first batches. This means to input all the batches during cross entropy evaluation, but use only those after the second one to calculate the result. It provided cross entropy boost of 0.003 ± 0.002 (not applied in Table 6).

4.3 SHAPE

An experiment was conducted to check how the distribution of hidden units in layers affect performance. Three shapes were defined (length ratios, in order from bottom to top, in parentheses):

RECTANGLE - all layer sizes are equal (1/1/1);

TRIANGLE - middle layer is twice longer than top one and bottom layer is 3 times longer than the top (3/2/1);

INVERTED TRIANGLE - middle layer is twice longer than bottom one and top layer is 3 times longer than the bottom (1/2/3).

The 3 shapes were trained using 4 different sizes of a 3 layer LSTM model. The particular sizes of each layer were chosen in order to follow the shape pattern and match the number of parameters in the rectangular networks. The same training parameters were used as in the previous section. All networks were trained only once. The results are presented in Table 3. The

Table 3: Test set cross entropy for different network shapes.

network	triangle	rectangle	inverted triangle
3LSTM64	1.616 (50/34/17)	**1.570** (37/37/37)	1.633 (19/38/57)
3LSTM128	1.382 (95/62/32)	1.370 (68/68/68)	**1.359** (34/68/102)
3LSTM256	1.230 (177/118/59)	**1.210** (126/126/126)	1.213 (62/123/185)
3LSTM512	1.116 (341/228/114)	**1.110** (241/241/241)	1.117 (116/233/350)

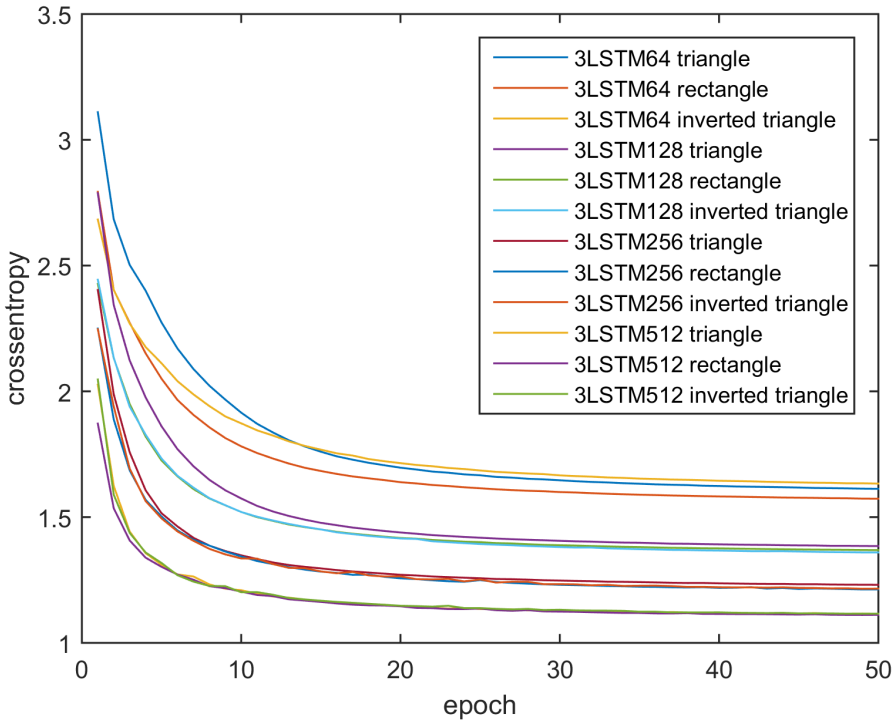


Figure 12: Validation set cross entropy plots for 3 different shapes and different model sizes.

table shows the test set cross entropy for the best validation result. Since the models were trained only for 50 epochs, it turns out to be the last epoch in all cases, except the 3LSTM256 rectangle, the 3LSTM256 inverted triangle and the 3LSTM512 rectangle, in which cases the best result was achieved in epoch 49.

As it is presented in the table, in general the rectangle architecture outperforms the other shapes. However, in case of the 3LSTM128, the inverted triangle brought lower cross entropy. The superiority of either the triangle or the inverted triangle shape varies with the size of the network and it cannot be stated which one performs better in general. However, the triangle shape did not improve the rectangle performance in any size category.

The validation set cross entropy is presented in Figure 12. It can be seen how close to each other in terms of performance these models are in each size category. A trend is noticeable, the bigger the model the smaller the differences between the given shapes. The biggest network seems to be almost invariant to the shape. It should also be noticed that in the beginning of the training, especially in case of the smaller models, the delivered cross entropy is substantially higher using the triangle shape.

The reason for the smaller models performing much worse in triangle and inverted triangle shapes is that they lower the number of dimensions

at one of the layers far below the size of the vocabulary (87). Therefore, their expression ability is reduced. In case of the bigger models the effect vanishes, since the size of the smallest layers is close to or even higher than the size of the vocabulary.

It was investigated if the inverted triangle 3LSTM64 learns a purely feedforward unit in the bottom layer. The purpose was to check if the network mitigates the issue of reduced dimensionality in the first layer by shifting some computations to the higher layers. The activations were analyzed on the whole validation set. A unit with the forget gate activation value below 0.3 for 74% of the time was found (the unit forgets most of its content for most of the time). Also the weights incoming to the cell of the unit were examined. The ratio of the mean absolute value of the input weights to the mean absolute value of the weights from the previous hidden state was 4.18. This means that the present input is responsible for 80% of the information flow in that unit. The input gate was closed (activation below 0.1) for only 9% of the validation set and the output gate for less than 1%. This unit is the closest one, found in the first layer of the inverted triangle 3LSTM64, to a purely feedforward unit. This unit indeed helps to shift the input, augmented in a limited manner, to the higher layers. For comparison, in the same sized rectangular shaped network the most extreme unit was keeping the forget gate below 0.3 for 65% of the time and the ratio of weights was only 2.5.

Throughout the rest of the thesis only the rectangular shape is used in all the experiments.

4.4 DEPTH

Different kinds of depth were evaluated as in Pascanu et al., 2013. The 3LSTM512 model was augmented with two dense layers in hidden-to-hidden transition and one dense layer in hidden-to-output. Standard tanh and ReLU nonlinearities were tested. LSTM with deep hidden-to-hidden transition had to be implemented, since the library did not support this feature. Sizes of the networks were chosen so in each layer, including deep transition and deep output, the number of units is the same and that the number of parameters is approximately of pure 3LSTM512.

After long time of cross-validating the parameters it turned out that models are extremely fragile to the learning rate and do not perform as well as expected. The best result was delivered by 3LSTM512 with deep transition and deep output with tanh, but the performance was much worse than pure 3LSTM512.

Three fixes to this situation have been proposed. First was to reduce the number of layers in between hidden states. Too many parameters were laying in hidden to hidden transition. Also the lower layers turned out to use more information from the input than from the hidden state (see Section 5.2), so the deep hidden transition was placed only in the top layer. Lastly, the gradient of the standard LSTM weights seemed very fragile to the hidden transition function, so two separate learning rates were introduced. One for standard LSTM weights and second, one order of magnitude lower, for deep transition and deep output connections. The idea was to train the hidden transition function slowly, in order not to interrupt the flow between hidden states too aggressively.

Table 4: Test set cross entropy for different deep networks. Deep hidden transition was applied only in the top layer. Abbreviations: DT - deep transition, DO - deep output.

Stacked layers	2		3	
Nonlinearity	tanh	ReLU	tanh	ReLU
Network				
DT	1.136	1.134	1.100	1.108
DO	1.124	1.128	1.117	1.119
DTDO	1.122	1.124	1.097	1.105
DT ReLU, DO tanh	1.107		1.096	
Stacked LSTM512	1.127		1.110	

The learning rates used were 0.006 and 0.0002, for standard LSTM weights and for dense layers, respectively. All other parameters stayed the same as in the previous experiments. Networks with 2 and 3 stacked layers were tested.

It was observed that in the case of 2LSTM512 with deep transition the ReLU units perform better and that the deep output delivered higher performance using the tanh nonlinearities. As a result, models with ReLU in DT (deep transition) and tanh in DO (deep output) were evaluated. They turned out to deliver even lower cross entropy, in both networks with 2 and 3 stacked layers.

Fourteen combinations in total were evaluated. The cross entropy results are presented in Table 4 together with pure (stacked) LSTM512 models for comparison.

It should be noticed that most of the cases with only a single augmentation, e.g. only deep transition in 2LSTM512, did not bring performance increase. However, applying both transition and output augmentations, with any combination of nonlinearities, always delivered better results. Also, it is clearly visible that the models with 3 stacked layers outperform networks with 2 layers.

5 VISUALIZING RNNS

5.1 ACTIVATION ANALYSIS

As in Karpathy et al., 2015, the activations of the units were analyzed. In the original paper only the cell state was presented. Since, according to Greff et al., 2015, the forget gate turned out to be the most important part of LSTM the corresponding forget gate activation is shown. Also the hidden output is presented for comparison. The activation values are presented as background color of each character separately on a part of validation set. All the units were taken from 3LSTM512.

The vast majority of the units are difficult to interpret. It looks like they pick up some features, but it seems impossible to make sense of them, as presented in Figure 13. A couple of interpretable examples are presented below.

PRINCE MOTIF

Some of the units seem to pick up motifs that are easier to understand. For example a cell that activates on various different characters, but it is the strongest on "Prince" sequence. It was found in the first layer of the network. It is presented in Figure 14. It turns from -1 on capital P into 1 on "rince". It also covers "Princess". This information might be useful for the network for the prediction of next characters. In the whole novel the most probable sequence starting with "Prince" is either "Prince Andrew" or "Princess Mary" (see Table 5 for counts). The hidden activation is weaker than the cell state, but still works in similar manner. The forget gate activation is low (cell forgets) on "Pr" and is high on "ince", but it also changes its value throughout the rest of the sequence.

POSITION IN LINE

Another unit found in the third layer turned out to keep track of the position in line. Its activations are presented in Figure 15. There is \r \n at the end of each line, two separate characters depicted as empty cells. The cell state and hidden activation change smoothly along the line, from -1 to 1 with 0 shifted towards the right end of the line. The cell state switches from (almost) 1 to -1 on \n at the end of the line. Hidden state goes to zero

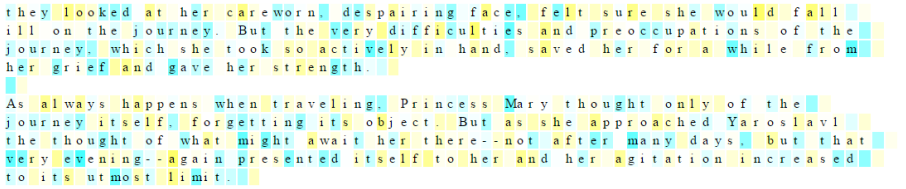
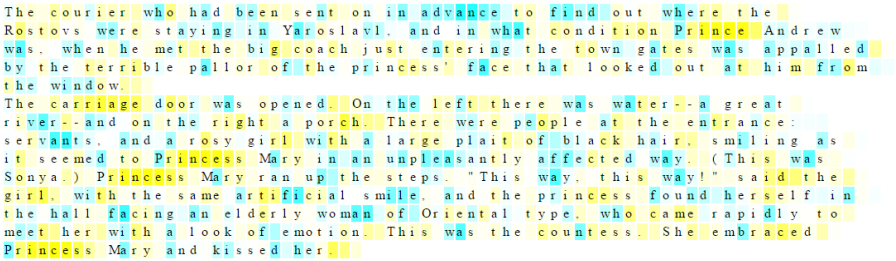
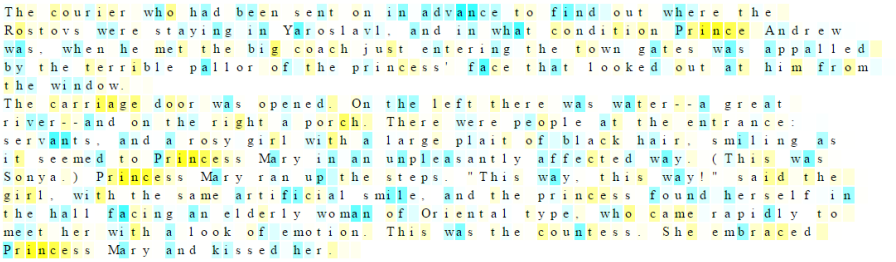


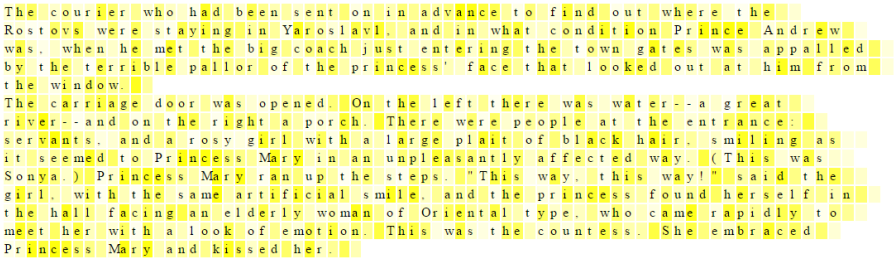
Figure 13: Cell state ($\tanh(c)$) of a randomly chosen unit that is difficult to interpret. Color represents the value of the activation, with yellow denoting 1, white 0 and blue -1. Presenetd on a part of the validation set.



(a) Cell state ($\tanh(c)$)

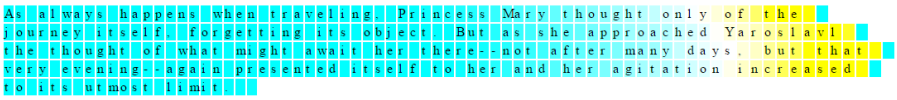


(b) Hidden activation

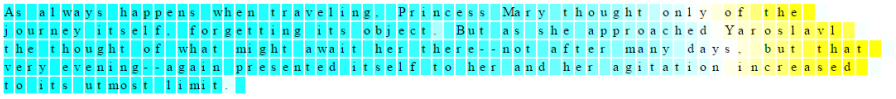


(c) Forget gate activation

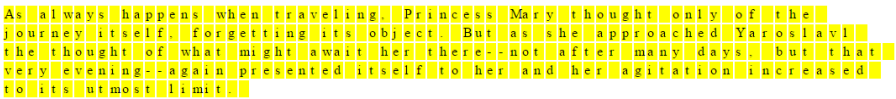
Figure 14: Activations of different parts of the unit that keeps track of the position in line. Color represents the value of the activation, with yellow denoting 1, white 0 and blue -1. Presenetd on a part of the validation set.



(a) Cell state ($\tanh(c)$)



(b) Hidden activation



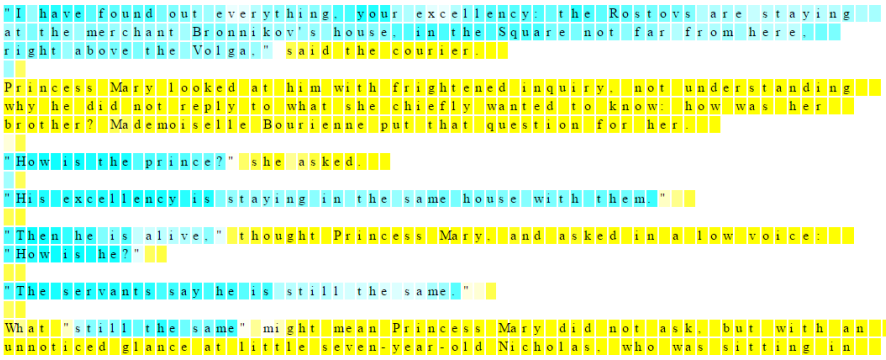
(c) Forget gate activation

Figure 15: Activations of different parts of the unit that keeps track of the position in line. Color represents the value of the activation, with yellow denoting 1, white 0 and blue -1. Presenetd on a part of the validation set.

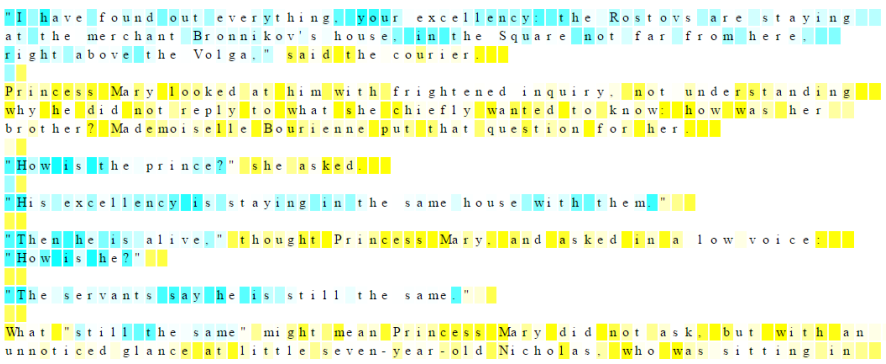
at \n. That is when the forget gate activation is zero (cell forgets) at \n - white color. The evolution of the unit is presented later in Section 5.5.

INSIDE QUOTES

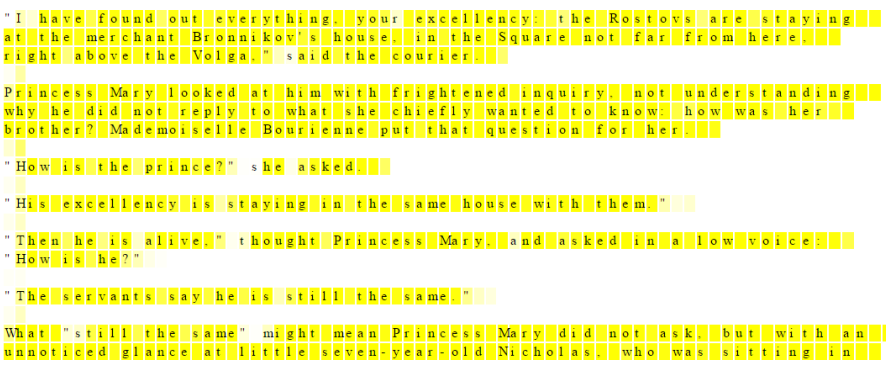
An interesting unit turns on inside quotes. It was found in the third layer. Its activations are shown in Figure 16. It can be seen that the cell state vividly extracts this feature. The hidden output also changes its sign inside the quotes, but it is not as strong. The mechanism of the unit is well explained by the forget gate, which activation is zero (the cell forgets) on quotes. An important finding is that the model is trained on backpropagation to 100 steps, but it correctly generalizes to more characters (see the first quote in the figure).



(a) Cell state ($tanh(c)$)



(b) Hidden activation



(c) Forget gate activation

Figure 16: Activations of different parts of the unit that turns on inside quotes. Color represents the value of the activation, with yellow denoting 1, white 0 and blue -1. Presenetd on a part of the validation set.

5.1.1 Cell state distribution

Similarly as in the supplementary material to Karpathy et al., 2015, cell state activations ($\tanh(c)$) were collected from the whole validation set and presented in cumulative histograms (Figure 17). The 3LSTM512 model was used. It can be seen that in the first layer the cell states activations form a distribution similar to Gaussian or Cauchy. In the second and third layer two extra modes are present in the histograms. The peaks are located at $\pm\tanh(1) \approx \pm 0.76$. When a unit forgets, the cell state is set to zero and the input is added, where ± 1 are the limiting values that can be added. That is the reason why the two extra peaks appear on the plots. It can be concluded that the units in the first layer do not forget their value often and/or their input gates do not saturate much.

As it is expressed by Karpathy, it also means that a cell cannot fully saturate in one time step after a reset. He proposed and evaluated an easy fix to that issue by augmenting the cell state equation with a parameter multiplying the input gate activation and cell input activation, so Equation 9 takes form

$$s_c^t = b_\phi^t s_c^{t-1} + \alpha b_\iota^t g(a_c^t), \tag{42}$$

where α is the parameter. Setting $\alpha = 2$ allowed the model to converge in 700 steps instead of 900. However, it is reported that the fix does not affect the performance after 50 epochs.

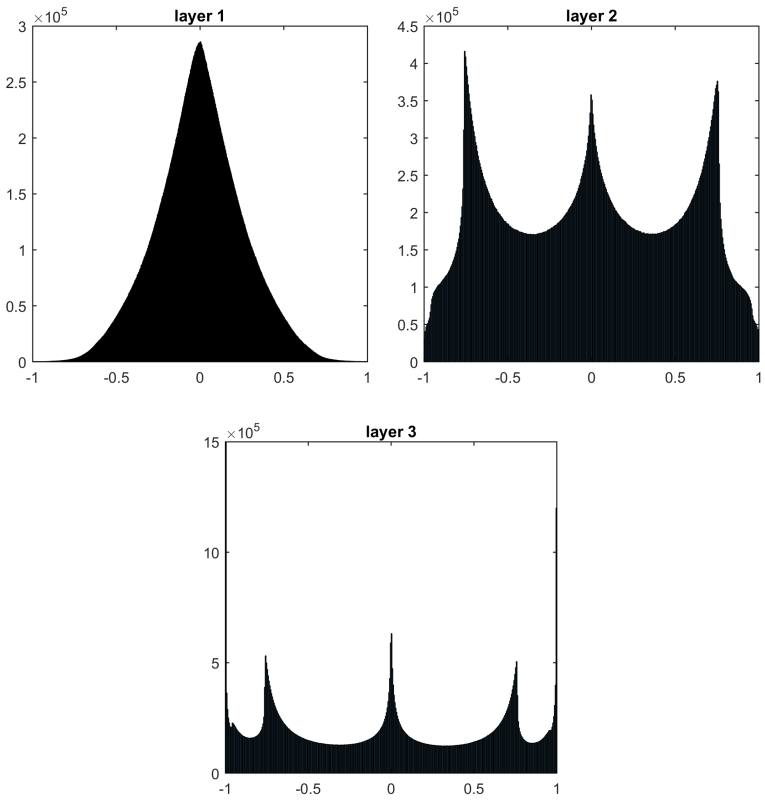


Figure 17: Cumulative histograms of cell state activations ($\tanh(c)$) for each layer separately.

5.1.2 Activation saturation

As in Karpathy et al., 2015, saturation plots of 3LSTM512 and 3GRU512 were prepared. The activation higher than 0.9 was defined as right saturation and activation smaller than 0.1 as left saturation. For each gate on each layer the fraction of times when the gate was left or right saturated was calculated. The results are presented in Figure 18 for LSTM model and in Figure 19 for GRU network. For all gates, the two saturation fractions must add up to maximally 1 - they always have to be below the black line.

There is quite a low amount of right saturated forget gates in LSTM model, which means that the majority of the units forget their values quite often. There is only a couple of units that purely integrate data over time. There are no cells with always left saturated forget gate, so none of the units serve as feed-forward connections.

There are a few units with almost always left saturated input gate, which means that the units produce their activations mainly based on the cell state and very rarely update it with the input. Most of the units on the top two layers have their input gate saturate at least in 40% of the time.

The majority of the units, especially in the third layer, have their output gate saturated for half of the time to either of the sides. Only one unit has

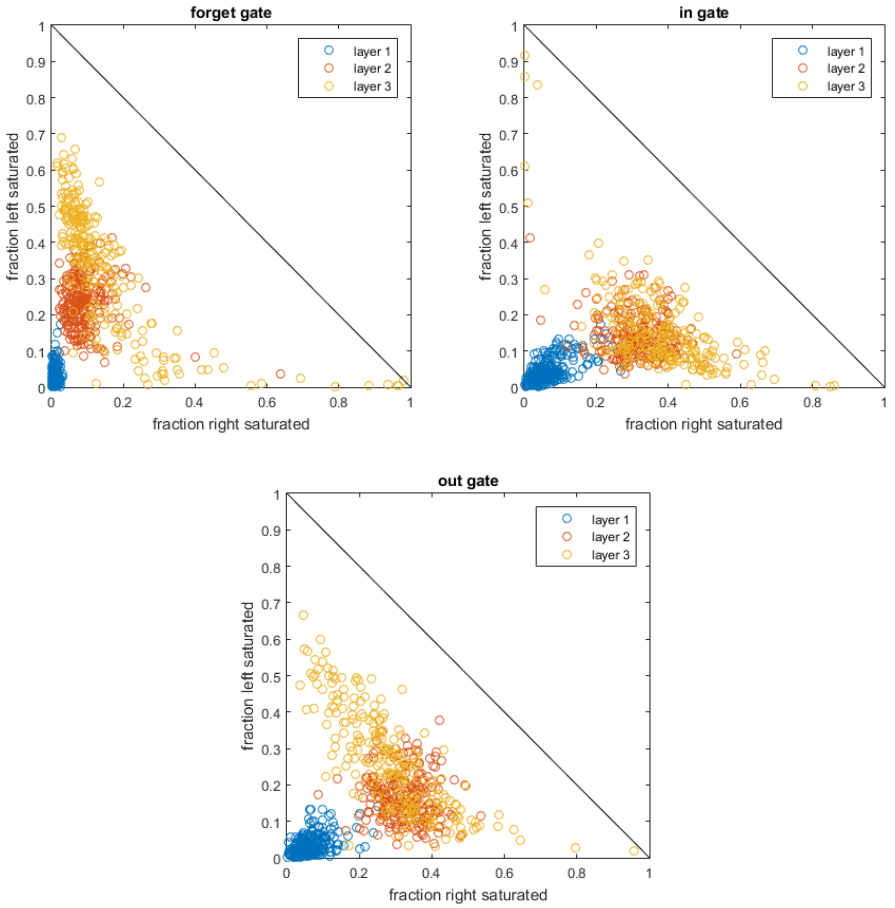


Figure 18: Gate activation saturation plots for 3LSTM512. Each cricle depicts a single gate. Its position is determined by right and left saturation fraction on the whole validation set.

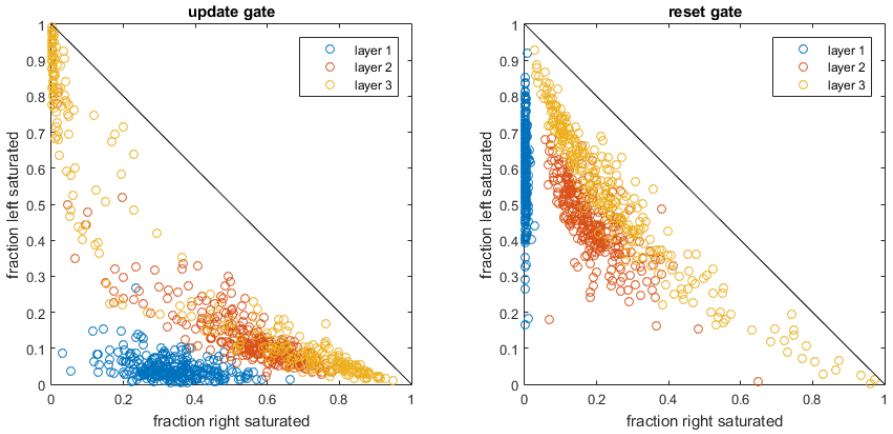


Figure 19: Gate activation saturation plots for 3GRU512. Each cricle depicts a single gate. Its position is determined by right and left saturation fraction on the whole validation set.

an almost always right saturated output gate. In such a situation the cell state is equivalent to the hidden state.

The phenomena of gates saturating very rarely in the first layer may indicate that they work as features amplifiers. They influence the flow of information by robustly scaling it.

There are many units in the third layer in the GRU model that have almost always right or left saturated update gate. They tend to either overwrite the state with the new one or keep the previous one.

The reset gate statistics shows that most of the units use more information from the input than the previous hidden states.

As it was visible with LSTM, GRU also behave differently in the first layer. The reset gate is never right saturated and quite often left saturated. This indicates that these units do not use much information about previous hidden states. Update gate statistics shows that the units prefer candidates activations over previous activation. This could mean that they also work as feature amplifiers.

5.2 WEIGHTS ANALYSIS

Weights of the network were analyzed. The 3LSTM512 and 3GRU512 trained on "War and Peace" were used. For each layer, for weights coming to cells and gates from input and previous hidden state separately, mean absolute values of weights were calculated and presented in Figure 20 and Figure 21, for LSTM and GRU model respectively.

In both models a phenomena is present that on the first layer the mean weight from hidden states is higher than from input. In higher layers the effect vanishes. In case of candidate activation in GRU the weights from previous hidden states even overgrow the input. This shows that in lower layers the models use more information from input and the higher the level the stronger the influence from hidden states. This finding motivated changes done in Section 4.4.

Each gate was also plotted using its mean absolute value of weights from input (x axis) and previous hidden state (y axis). They are shown in Figure 22 for LSTM model and Figure 23 for GRU network. All gate weights

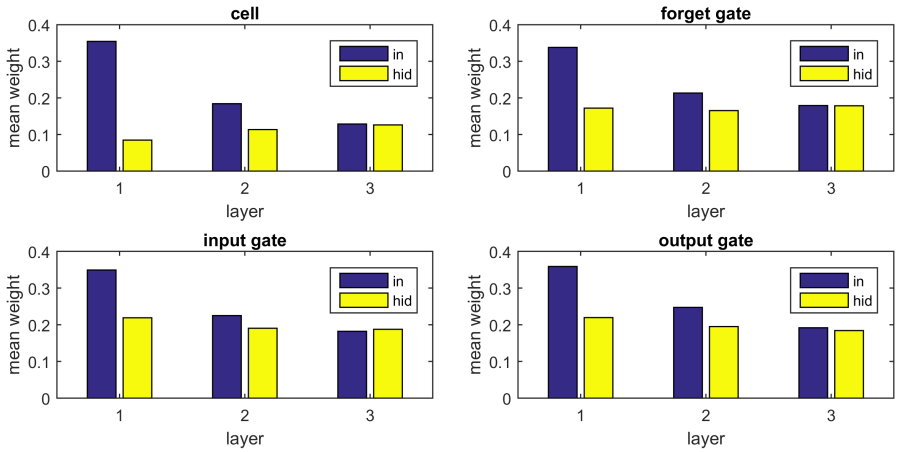


Figure 20: Mean absolute values of weights for each layer of 3LSTM512. The weights are presented for coming from input (blue) and previous hidden state (yellow). Plots show weights incoming to cell and each gate separately.

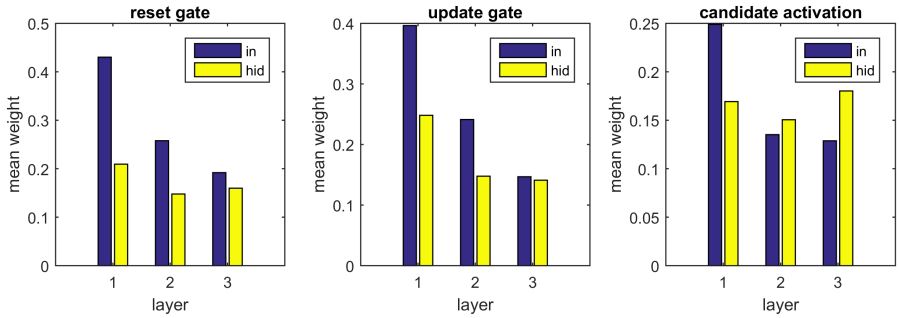


Figure 21: Mean absolute values of weights for each layer of 3GRU512. The weights are presented for coming from input (blue) and previous hidden state (yellow). Plots show weights incoming to cell and each gate separately.

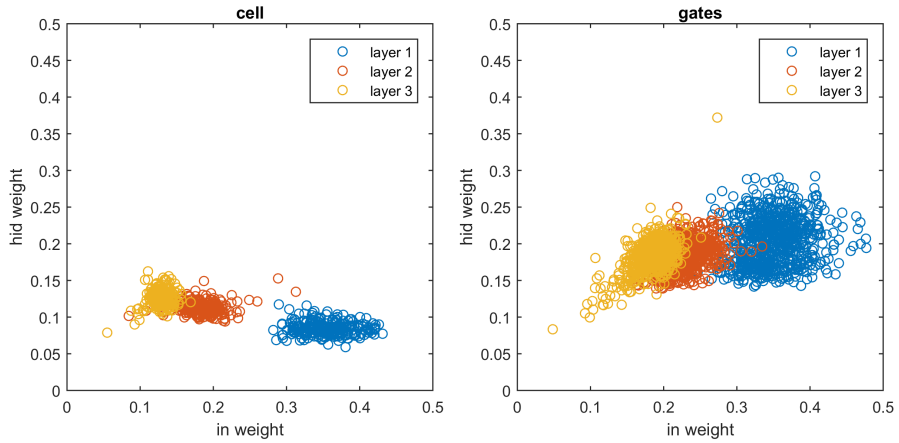


Figure 22: Weight plot of 3LSTM512. Each circle represents a cell or a gate. The position of circles is determined by incoming weights from input (x axis) and previous hidden state (y axis).

are presented on one subplot, since the purpose was to separate them depending on their function - main information flow and gating mechanism.

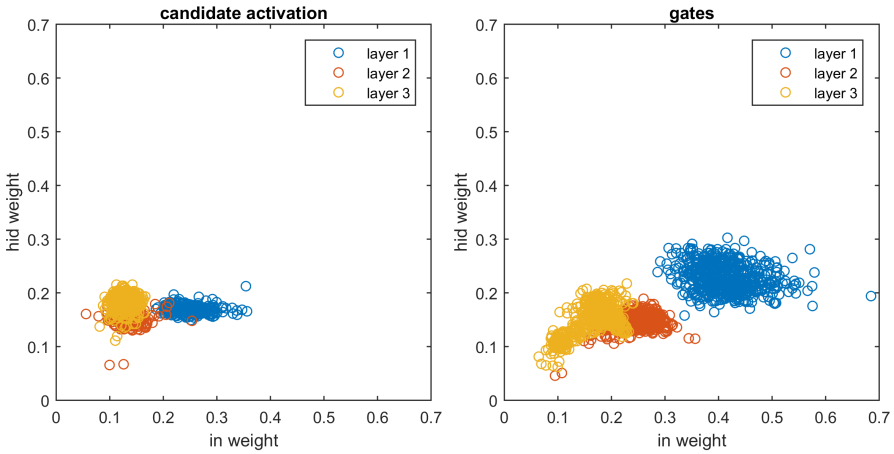


Figure 23: Weight plot of 3GRU512. Each circle represents a cell or a gate. The position of circles is determined by incoming weights from input (x axis) and previous hidden state (y axis).

The phenomena of decreasing input importance in the higher layers is also present on the weight plots. They also show that the gating mechanism uses more information from hidden states than the main flow, especially in case of LSTM. The effect is clear only in the first layer of GRU.

5.3 SAMPLING

Similarly as in Karpathy, 2015, an experiment was conducted to see "what does a neural network think" on example of the 3LSTM512 model trained on "War and Peace" dataset. First, a random capital letter was picked with uniform distribution. Then, it was inputted to the network. The produced probabilities ($Q_t(i)$) were processed using augmented soft-max function:

$$P_t(s) = \frac{\exp(Q_t(s)/\tau)}{\sum_{i=1}^{n} \exp(Q_t(i)/\tau)} \tag{43}$$

to get new sampling probabilities, for each symbol s out of n symbols in vocabulary. The τ parameter (temperature) controls how likely it is to sample a symbol different the the most probable one. This process was repeated multiple times to get a whole sequence.

For τ very close to zero (choosing the most probable character) the output was very monothematic:

What has
the same time the count was a man who has been so strongly and
sorrowful and so much as to the countess."

"What do you think of the countess?" said Pierre.

"What do you think of the countess?" said Pierre.

"What do you think of the countess?" said Pierre.

"What do you think of the countess?" said Pierre.

"What do you think of the countess?" said Pierre.

"What do you think of the countess?" said Pierre.

"What do you think of the countess?" said Pierre.

"What do y

At the same time it is the most probable sequence starting with "W" according to the model trained on "War and Peace" dataset.

Allowing the model to choose also the less likely characters ($\tau = 2e-2$) results in more interesting sentences:

The lady
was a strange smile of the soldiers who were silent.

"What is the same time?" said the countess.

"What a strange son can read the countess?" said the countess, and with
a smile of horses and Anna Pavlovna said to himself. "I will tell you
that I am so strong to him and that is the same time to me to the princess
and the honor of the regiment with the same time the countess was as
if the troops were standing before him and the same place where the
countess was saying

When the temperature was pushed even higher ($\tau = 5e-2$) the model started to make spelling mistakes:

Natasha
6w she was srepting him and only made xact questions and s, and the
*8ther 8fter 84ze and amused him and the * which he had Russians and*
Schon Grabern began to EQuete of men and Mademoiselle Bourienne and
the more serious reOVeranski was Emperor under the contrary to a Russian
żNvites and TZ(vikhin8 he 5) were -and Denisov
known that he was to find the first time that ; the
count 1as not 8nd , considered men who I never Orders to Question Moscow
and (the victory Ăld princess ? Vas

It should be noted that the all the hidden and cell states are initialized to zero before sampling. In case of the unit that represents the position in line (Section 5.1) it means that the current position is close to the end, one or two words more will fit (see Figure 15). This phenomena can be seen in all presented samples.

5.4 CONTEXT MEMORY

An important feature of a language model is its ability to keep information about the context. In case of RNN it means that the information about the past characters in the sequence should be preserved in the last hidden state. It was already shown that the model stores some information, i.a. opening and closing the quotes. Let us check if it does remember less explicit information.

Few experiments were designed to investigate it. In all of them "War and Peace" dataset was used. In the first one, each one of seven arbitrarily chosen sentences was concatenated with each available character giving 609 new combinations. All of them were separately inputted to 3LSTM512 with hidden states initialized to zero. The hidden states of the last characters were saved.

The t-SNE plot shows that sequences with the same sentences in the beginning are similar. It is presented in Figure 24. It should be noticed that some punctuation marks (?, ., :, !) form another group on the plot. It shows that these characters make most of the units forget gates closed (make them forget previous information). It lowers the confidence of the model of what should be the next prediction, which seems correct taking into consideration what characters are they - practically any word could start afterwards.

When experiment was repeated with two exactly the same characters added after the sentence the t-SNE plot did not show any resemblance between the sentences, only between the last characters. It was also the case if instead of hidden states the forget gate status was used.

In another experiment, each one of seven sentences was concatenated with each word from a set of six extra words giving 42 new sequences. All of

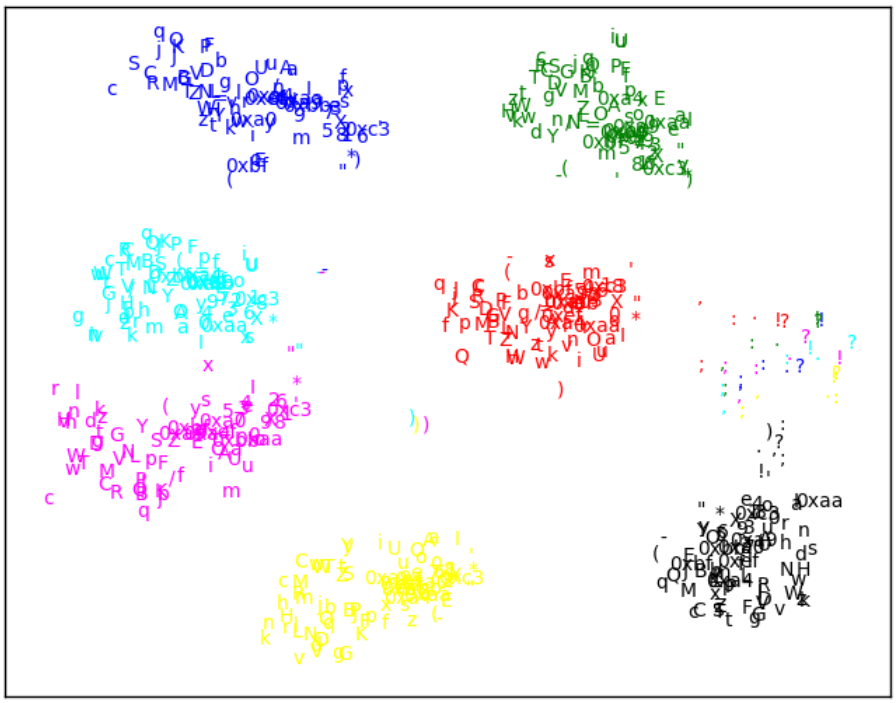


Figure 24: t-SNE plot of representations of sentences finished with different character.

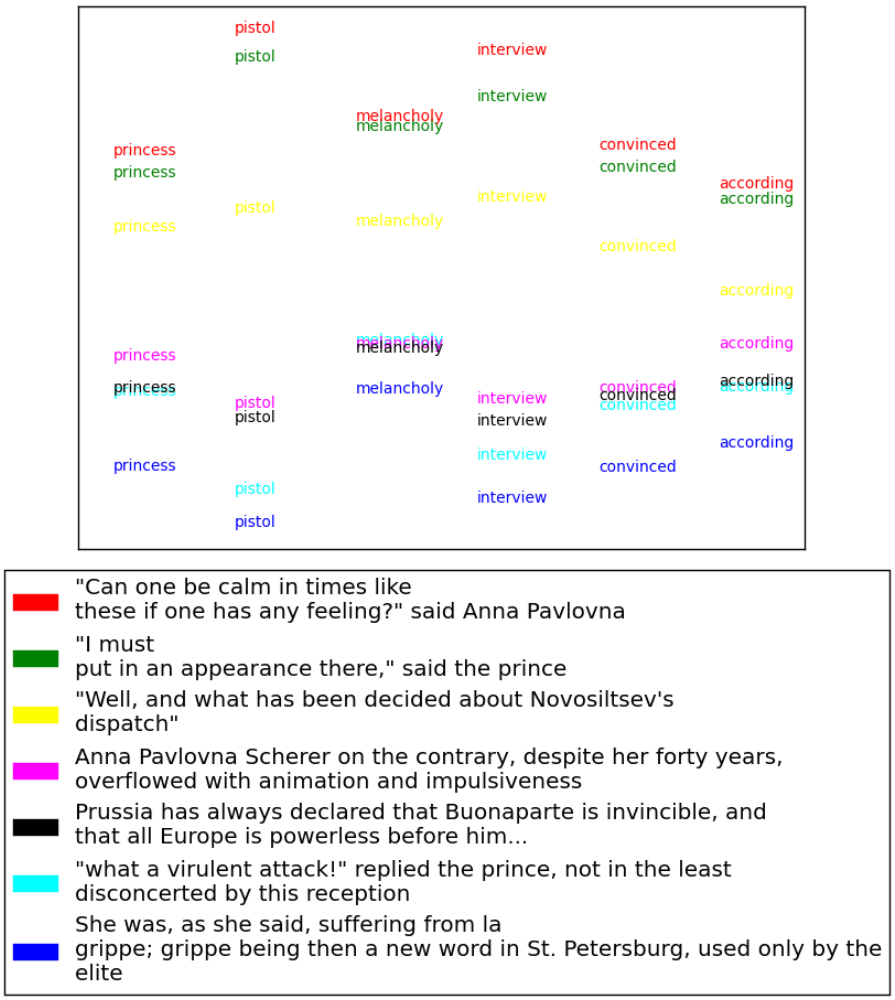


Figure 25: Sixth component of PCA (y-axis) of hidden states after inputting sentences ended with an additional word.

them were separately inputted to 3LSTM512 with hidden states initialized to zero. The hidden activations of the last characters were saved.

The t-SNE plot of the hidden states did not present any resemblance between the original sentences. The amount of information about the context stored in the hidden states was too small.

On the other hand, Principal Component Analysis (PCA) of these hidden states delivered few components, but the most interesting was the 6th one, which seems to distinguish between the sequences based on the original sequence. The component is responsible for 4.2% of the variance. This shows that the model, in a rather limited manner, keeps track of the context information. The plot of the component is presented in Figure 25. The y axis represents the value of the component. The extra words are printed on the figure instead of points. The words are spread evenly through the x axis. The color of the word represents the original sentence, as presented in the legend.

The same experiment, but with exploiting cell states instead of hidden activations, brought similar results. However, they were not as clear.

5.4.1 Memory units

In an attempt to further evaluate the network's ability to remember the context another experiment was conducted. On the whole validation set, units that keep information for some time were extracted. It was achieved by picking only the units that keep their forget gate activation above 0.4 for at least 20 time steps in a row at least 200 times, which equals to 0.0625% of the time. A set of 42 units located in the third layer was obtained. It confirms the earlier finding that the higher layers use more information from the previous hidden states compared to the lower ones (Section 5.2).

Experiment with sentences and words was conducted again, using cell activations only from the set of the chosen *memory units*. Also a new set of 500 extra words of length between 10 and 15 was used. A t-SNE plot of the cell activations separated the words according to the starting sentences quite well. However, some of the clusters were overlapping.

To address this issue, cell states were plotted for a set of chosen sentences and words, for each one of the 42 units. It was discovered that some of the cell states alone are able to separate the sentences correctly. Figure 26 presents 15 words with their position on y axis controlled by the cell state of one particular unit after inputting the whole sequences. Each unique word is spread evenly on the x axis.

Three units that separated the data most clearly were chosen. A t-SNE plot of their cell states is presented in Figure 27. The legend from Figure 25 applies. It can be noticed that the sequences starting with the same sentence form separate clusters in the t-SNE plot. The set of 3 units corresponds to 0.41% of the number of all units in the three layers of 3LSTM512. These units carried enough information to distinguish between the starting sentences with quite high accuracy. This experiment shows that the presented model indeed keeps some information from the past sequences.

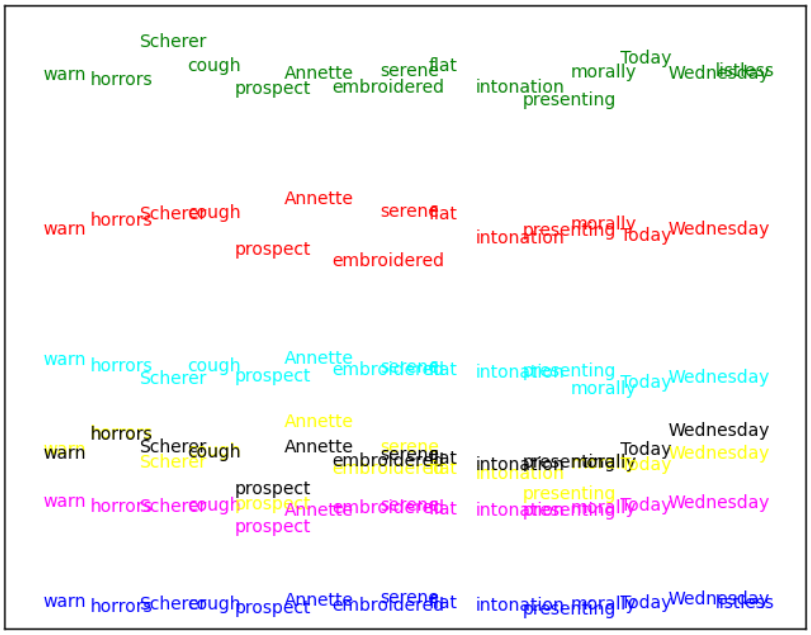


Figure 26: Cell state (y axis) of the unit that turns on slightly inside quotes and outside of them, after inputting sentences with appended words.

Figure 27: t-SNE plot of cell activations of the three *memory units*. The legend from Figure 25 applies.

The three *memory units* were further inspected. The first one represents the position in line. The second unit turns on inside quotes. These are the exact same units identified in Section 5.1. The last unit, not presented yet, slightly turns on inside quotes, but also at some positions outside, where it is difficult to interpret. The last unit is the one depicted in Figure 26.

In the t-SNE plot, the green and red sentences overlap at times. The reason, most probably, lays in the fact that they end with "said the prince" and "said Anna Pavlovna", respectively, which are rather similar. They are well separated by the unit from Figure 26, but the two other units do not work as well in this case.

However, the two other units were necessary, since they help to distinguish between the pink, black and yellow sentences, which the first unit does not separate well. The unit sensitive to the position in line helps distinguish the most the cyan sentence, since the length of the last line of that sentence is the most unique. It also separates the red, green, black and pink (long last lines) from blue and yellow (short last lines).

The clear inside quote unit separates two kinds of sentences. The ones with a quotation inside followed by text. And the sentences, in which quotation marks do not appear or sentences starting and finishing with a quotation mark. Analogous plots to the one in Figure 26 for the other two units are presented in Appendix C.

Even though the unit representing the position in line appeared to enhance the performance in the conducted experiment, it does not provide much information about neither the structure of the words nor the motifs in the sequences of characters. To mitigate this issue, another set of units was chosen for the sequences representations. In the original list of 42 *memory units*, two more units were found that were activating fuzzily inside the quotes and on some difficult to understand patterns outside of them. At the same time, they were providing some information that helps separate the sentences.

Three more units that were capable of distinguishing between some of the sentences were localized. Their interpretation is rather difficult. One of these units was activating on some chosen spaces, colons, periods, commas, exclamation marks and a few more characters. The second unit was changing its activation on "Princess Mary", "Yaroslavl", "excellency", on french translation and in other random-looking places. The last one was firing more clearly on "Princess Mary", "Yaroslavl", "Prince Andrew" and a couple of different words. They are all presented in Appendix C, in the same manner as in Figure 26 and also as background color on a part of validation set, as in Figure 15.

Similar experiment as the previous one was conducted, but this time the seven introduced units were used. The unit representing the position in line was removed, the two new fuzzy inside quotes units and the three difficult to interpret cells were added. The resulting t-SNE plot of the augmented sentences representations is depicted in Figure 28.

It should be noticed that the sentences are very well separated, even without the usage of the line position unit. It was achieved using a set of inside quotes and seemingly chaotic units. Since they are difficult to interpret, it seems impossible to draw concrete conclusions. However, it can be stated that the network learned seven units that keep track of the past sequences, based on the relations between the consecutive characters. It is an important statement, since it means that the model indeed has a context memory.

Figure 28: t-SNE plot of cell activations of the seven *memory units*. The legend from Figure 25 applies.

5.5 TRAINING DYNAMICS

An experiment was designed to empirically check how do the predictions made by the model change during the training. The 3LSTM512 model with "War and Peace" dataset was used. Four different categories were evaluated.

PROPER NOUNS

For the sake of the experiment, one or two words long expressions that appear in an unchanged form throughout the novel with some frequency will be called proper nouns. Inputting a couple of first characters of a proper noun should predict the rest, since it is the most probable prediction. Different proper nouns were tested, as presented in Table 5 together with the number of occurrences. All proper nouns were chosen arbitrarily. When there is more than one expression starting with the same word, the most common one in the novel should be predicted. It was also tested how the model will cope with different cutting points, e.g. input "Princess " should predict "Mary", as it is the most likely, but "Princess D" should give "rubetskaya".

Figure 29 depicts the points, after which the prediction for given proper noun is correct until the end of training. It can be seen that the expressions that appear often in the novel are learned very early, with "Princess Mary" and "Prince Andrew" only after the first epoch. The trend of learning less frequent expressions later is noticeable.

The similarity between proper nouns also plays role, as in "Princess Bolkonskaya" and "Princess Drubetskaya". They are both extremely rare and similar. The first one appears in three distant places in the book. The second 5 times in the beginning of the novel and once in the middle. It explains why it took model so much time to learn it. It seems unreasonable that "Princess Kuragina" is used only twice in the whole book, but it is discovered earlier than "Bolkonskaya" or "Drubetskaya". The reason is probably that the last two are extremely similar to each other. The first occurrences of "Princess Bolkonskaya" and "Princess Drubetskaya" being properly predicted are epochs 22 and 37, respectively. However, the model loses this features from time to time until the points presented in the figure.

Since "Prince Vasili" differs much from the other prince names and it appears quite frequently in the novel it is discovered already after the second epoch. "Prince Bolkonski" and "Prince Bagration" share the same first letter. According to their frequencies "a" should be predicted after "B". It is the case after epochs 3 to 9 and 12 to 50. However, after epochs 10 and 11 the model predicts "o". It is probably due to the fact that "Prince Bagration" appears

Table 5: Number of occurrences of chosen proper nouns in the "War and Peace" novel. The count is also presented as a fraction of all words in the dataset in permilles.

proper noun	count	fraction	proper noun	count	fraction
Prince Andrew	981	1.74‰	Princess Mary	487	0.86‰
Prince Vasili	206	0.37‰	Princess Bolkonskaya	7	0.01‰
Prince Bagration	43	0.05‰	Princess Drubetskaya	6	0.01‰
Prince Bolkonski	28	0.04‰	Princess Kuragina	2	<0.01‰
Count Rostopchin	44	0.08‰	Anna Pavlovna	142	0.25‰
Count Rostov	29	0.05‰	Anna Mikhaylovna	115	0.2‰
Count Bezukhov	25	0.04‰	Moscow	722	1.28‰

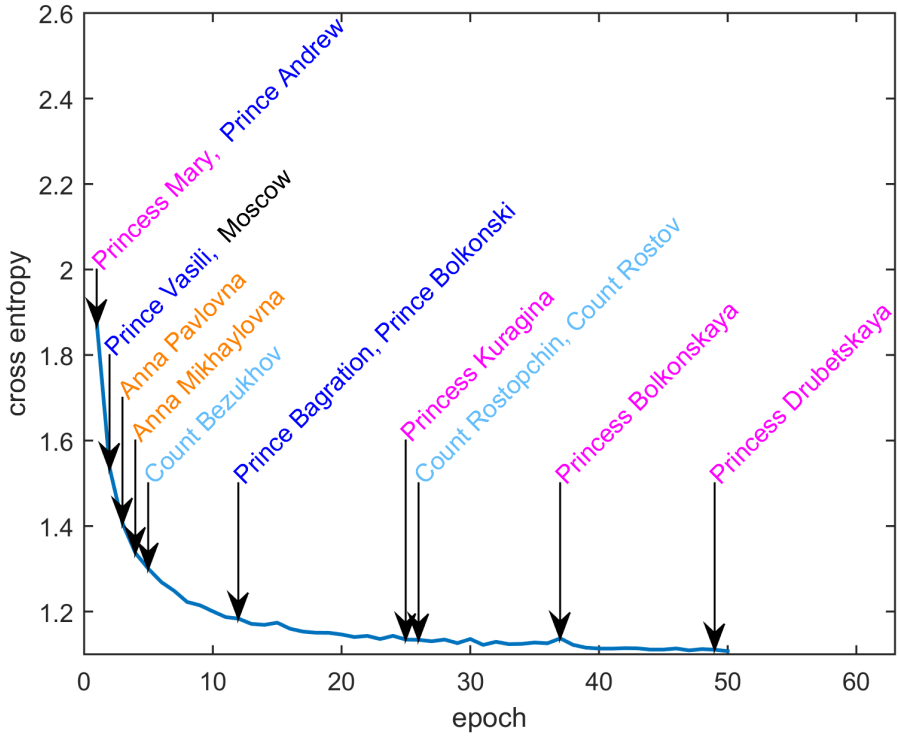


Figure 29: Dynamics of *proper nouns* presented on the validation set cross entropy plot of 3LSTM512. The arrows point at epochs, after which the nouns are predicted properly. The color of the expression is based on the first word.

in 4 distant places in the novel and occurrences of "Prince Bolkonski" are spread more through the book.

"Anna Pavlovna" is more frequent than "Anna Mikhaylovna", so the models learns it one epoch earlier. The two proper nouns are quite different, so the network easily distinguishes between them.

"Moscow" is broken into parts as "Mos" and "cow". It is learned after the second epoch, even though it is more frequent than "Princess Mary" discovered after the first epoch. The reason might be the fact that "Moscow" is much shorter, so it carries less information that can be used for prediction.

"Count Rostov" is exactly the same as "Count Rostopchin" up to the last character. Since "Rostopchin" is more frequent, the network should predict "p" after "Prince Rosto" instead of "v". However, at first the model prefers the "v", it switches throughout the training from one solution to the other to settle with more frequent "p" after epoch 26. Since "Count Bezukhov" differs from the other two, it is learned after fifth epoch.

FINISHING A QUOTE

After inputting a sentence starting with a quote and ending with a question mark the network should predict a quotation mark. If the sentence is too short it also could be a space or a newline. Two input sequences were tested: long - *"Seven days had passed. What were you saying to him?* and short - *"Seven days had passed. How are you?*.

The results are presented in Figure 30, where validation set cross entropy is plotted. The color of the line represents the predicted character. It can be seen that quotation mark is the most common ending to both sequences and that is how it stays after training. Both long and short sentence are more

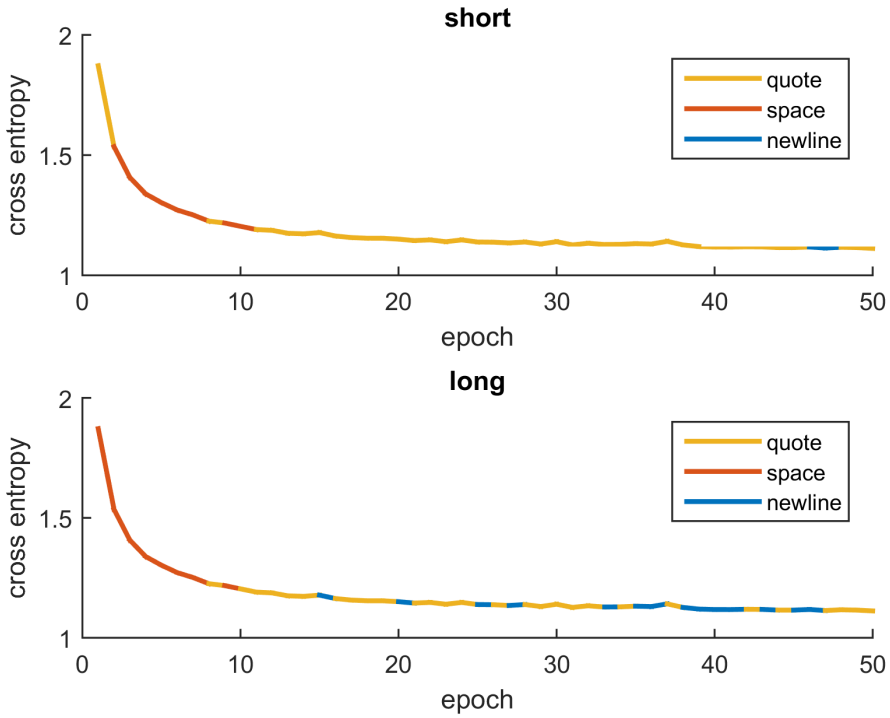


Figure 30: Dynamics of *finishing a quote* presented on the validation set cross entropy plot of 3LSTM512. The color of the plot represents prediction of the next character.

likely ended with a space in the beginning of training. The difference lays in the fact that the model tends to end longer sequence with newline more often. This is most probably due to the fact that the long sequence is 50 characters long - it is very close to the maximum length of the line.

ANSWER WITH QUOTES

After inputting a question in quotes on both sides the most likely prediction should include another statement in quotes. The prediction is enrolled up to 300 characters in search for two quotation marks. Two sequences were evaluated: *"How are you?"\r\n* and *"How are you?"*. The results were identical for both of them. The results are presented in Figure 31. After the sixth epoch the model correctly predicts a sequence including two quotation marks: *"I am a strange and the same thing is not to say that I was a long time to the countess.\r\n\r\n"*, even though it does that after two newlines in order to start another quotation. After the next epoch it closes the quote correctly. The model keeps learning and forgetting how to predict an answer in quotation marks and after 35th epoch the network finally learns the ability and does not lose it again. The answer evolves into *"What do you want?"* in epoch 35 and to a profound *"Why do you say that?"* in epoch 50. All intermediate answers are listed in Appendix A.

NEW LINE

After inputting a capital W the network is supposed to make at least 2 newlines (\r\n) in 150 characters (the longest line in the novel is 74). The model learns the feature after 9 epochs and never loses it afterwards. It is the moment when the unit representing the position in line is developed, as presented in Figure 32.

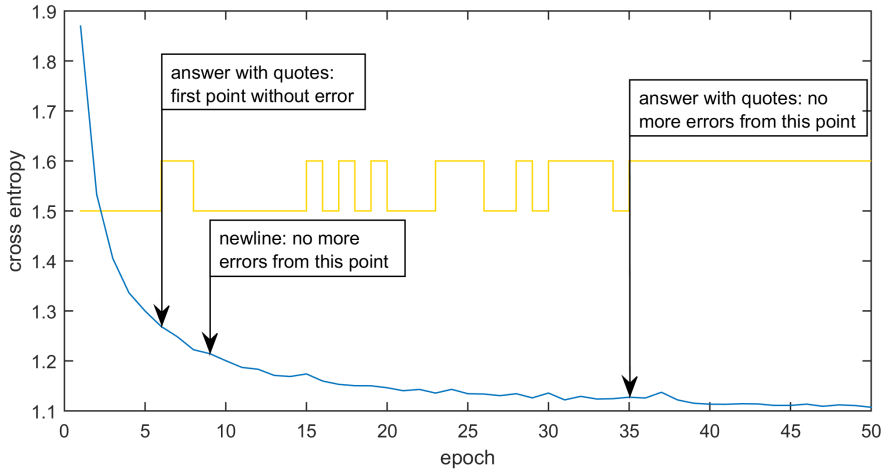


Figure 31: Dynamics of *answer with quotes* and *newline* presented on the validation set cross entropy plot (blue) of 3LSTM512. The yellow line represents the *answer with quotes* error (low - incorrect, high - correct prediction).

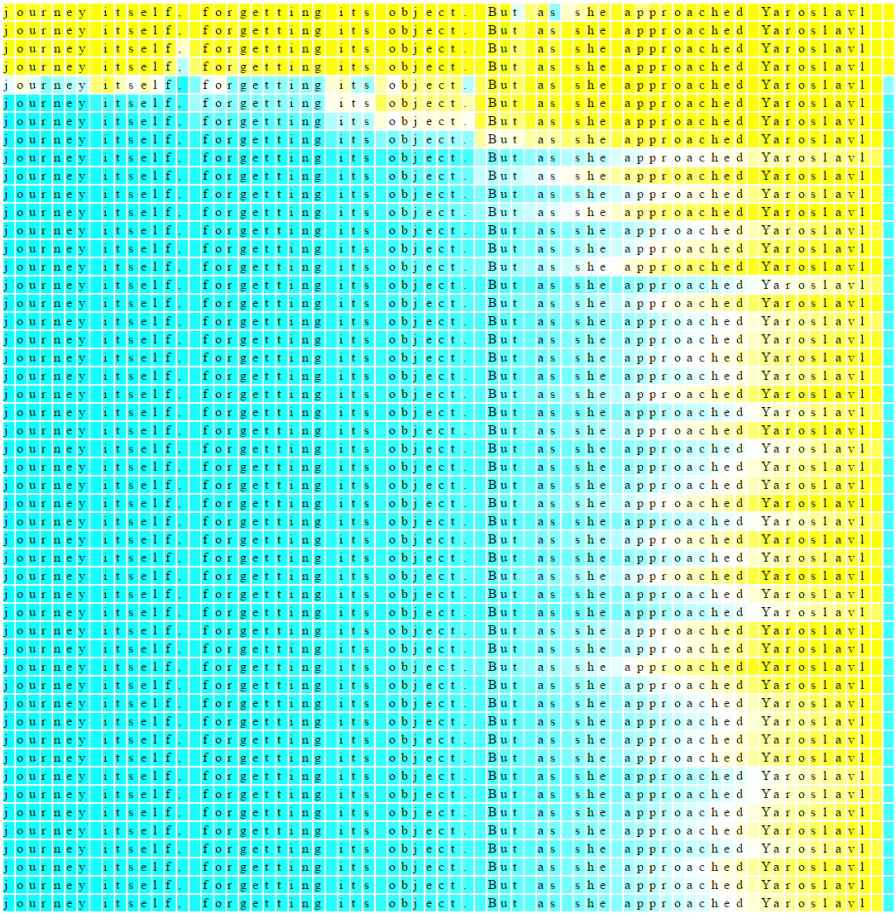


Figure 32: Evolution of unit representing position in line (as in Section 5.1). Each line depicts cell activations after each epoch of training, starting from the top. The background color of the characters show the value of the activation at that point of sequence, with yellow denoting 1, white 0 and blue -1.

5.6 WORD REPRESENTATIONS

In order to check how does the model creates word representations an experiment was conducted. All words of length at least 10 from "War and Peace" dataset were inputted separately to zero-initialized 3LSTM512 and the last hidden activations were stored and plotted using t-SNE. It is presented in Figure 33. It should be noticed that the representations are highly dependent on the word ending. Suffixes of the bigger clusters were listed and colored (in case of words that end with two of chosen suffixes - it is presented as the longer one). The experiment repeated with cell states instead of hidden activations delivered almost exact same results.

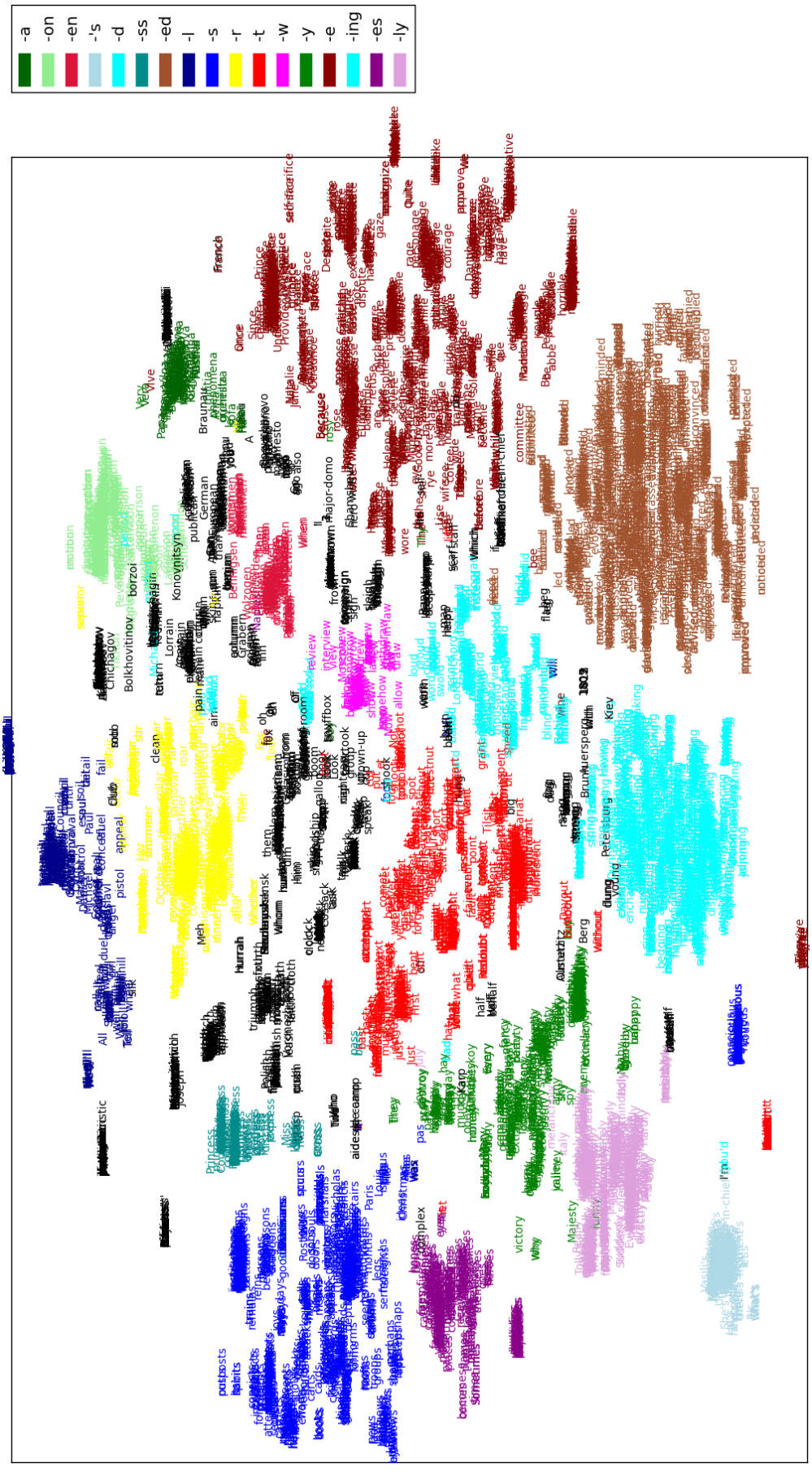


Figure 33: t-SNE plot of words representations.

6 CHAR-WORD ARCHITECTURE

As it has been presented, the character level model does learn structure of the text. It is able to find quotes and even some more ambiguous motifs. But it seems to learn the representations of the words based on their structure (e.g. the -ing suffix) rather than representation of its role in the sentence or the meaning (in Cho et al., 2014 similar words in function were close to each other on a t-SNE plot - presented in Figure 34).

In order to mitigate this issue, another architecture was designed. The goal was to train a network to find representations of both the words and context and utilize this information to predict the next word.

The proposed architecture is presented in Figure 35. A word is inputted to the network character by character with an end-of-word token (\e). When the network steps on that token it stops reading the input and starts predicting the next word, also finished with \e.

The augmented 3LSTM256 model was validated with many different configurations of hyper parameters, but it did not deliver any good results. It turned out that it is far more difficult for the network to learn to predict in this configuration than in the previously presented character-level model. After all, in such an architecture the network has to read all the characters,

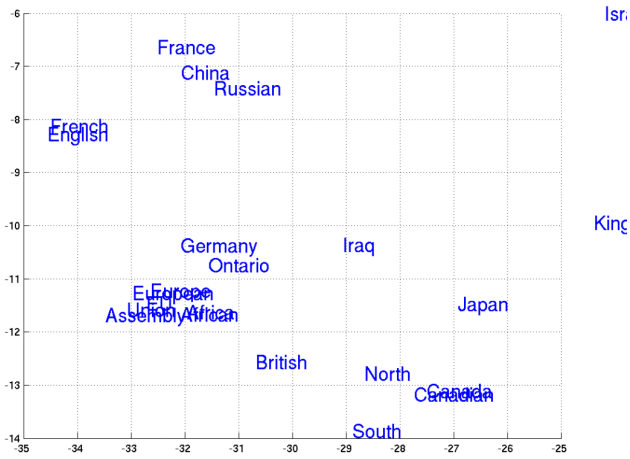


Figure 34: Part of a t-SNE plot of the words representations learned by the RNN architecture in Cho et al., 2014.

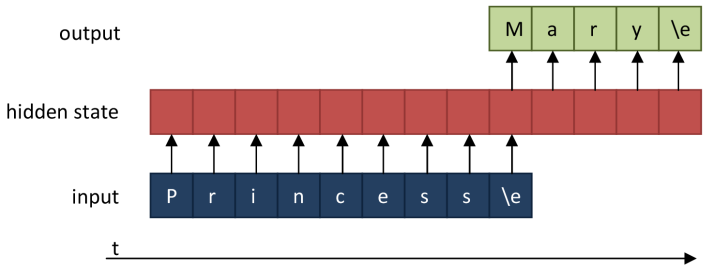


Figure 35: Schematic of proposed char-word architecture.

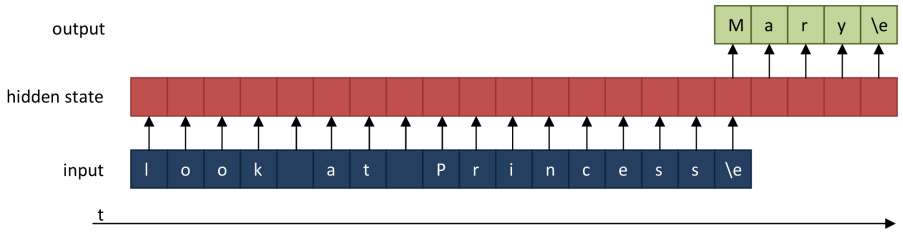


Figure 36: Schematic of the augmented char-word architecture.

somehow store them in the hidden activations and then utilize the collected information for prediction. The model was predicting \r\n quite often, since it is a common sequence appearing about every 10 words.

During training, the hidden states after inputting one word were stored and used in the beginning of the next word. So the gradient was calculated around the correct point, but it was stopped from backpropagating to the previous characters. The propagation path was too short.

The architecture was extended to address this problem. It is presented in Figure 36. Using this solution the gradient could backpropagate up to 100 characters, which has been proven to be enough in the character-level models.

Much effort was put to validate the hyper parameters and train the model. However, the lowest cross entropy value on the validation set achieved was 2.43 after 50 epochs, which took two days to compute (for comparison training the most complicated model from the previous chapters took less than 7 hours on the same computer). The performance was increasing with every epoch and the model seemed to be really far from convergence, but the trend was forecasting a cross entropy value of approximately 2 at convergence. It was concluded that the model is lacking the expression ability.

The network was changed to the 2LSTM512 with deep transition (ReLU) in the top layer and deep output (tanh), as it was presented in Section 4.4. The weights from the first layer of the already trained character-level model were transferred. After the first epoch the network already reached 2.55 cross entropy on the validation set and it outperformed the previous model after only 6 epochs. Furthermore, it was mostly predicting correct simple English words like "the", "an", "The" (after a period and a newline), "had", "see" (i.e. in a place, where "means" was the target), "too", "so", "was", "that" and "\r\n".

This shows that this architecture has a potential, but it may be necessary to use even a bigger network in order to reach the performance of the simple character-level model. Furthermore, the estimated time to train the model for 50 epochs on the machine used was four and a half days. It makes the model less feasible than initially expected.

Assuming the network was trained, a huge advantage of such a model would be the power of expression. The representations would be much richer. Since the network has to read all the input and memorize it, there would be plenty of space for finding dependencies that would be impossible to find by a simple character-level model.

Compared to word-level models the architecture is also beneficial. Instead of using vocabulary of words the text can be inputted character by character. The unknown words would be handled naturally. Also, the network could possibly try to learn them in real time.

7 CONCLUSIONS

The main goal of this thesis was to understand Recurrent Neural Networks. It was achieved by providing the theory behind them and evaluating the presented concepts in experiments.

The models presented in the thesis have proven their performance and expression abilities. Results from a very recent paper were reproduced. Achieved performance is lower, but it scales similarly. Parameters fine tuning should bring lower cross entropy values in reproduction. However, wide range of knowledge required to research Recurrent Neural Networks and time needed for training the models prevented from doing so in this master project.

Even though there are various methods for training the Recurrent Neural Networks available and researchers have been developing them for many years, it is still a challenging task to fine tune models efficiently. There does not exists one best method for training networks. Training takes plenty of time and uses enormous amounts of memory. The trend in computational power of computers draws an optimistic forecast for the future, but the training algorithms must be excelled as well. A method that does not need any tuning would bring a huge performance boost, since one would not waste time on cross validating different parameters.

The experiment with distribution of units in layers (shape) showed that it is mostly beneficial to put the same amount of units in all layers. In the first layer of the 3LSTM64 inverted triangle, a unit was localized that was passing the input to the higher layers without many changes. Even though it was not in a purely feedforward fashion, the task of the unit was more clear than in the same network in rectangle shape.

LSTM networks with deep transition and deep output delivered indeed better results than the pure stacked networks. It was achieved by using two learning rates, separately for the LSTM and FFN connections and by applying the deep transition only in the top layer. The ability to express the hidden-to-hidden transition with much more complicated function provided lower cross entropy values. However, the best result from such networks was still worse than the best Karpathy's result without using these concepts. Again, fine tuning the parameters should mitigate this issue.

It was shown that the units in the models learn to extract different features and some of them are interpretable. A highly important phenomenon was presented that the quotes are learned on backpropagation to 100 steps, but they easily generalize to more. It justifies that a truncated backpropagation harms performance only in a limited manner.

Activation and weight analysis showed that the lower layers utilize more information from the input than from the previous hidden states and the opposite in the higher layers. It allowed to improve the deep transition models by implementing it only in the top layer.

It was also shown that the gating mechanism uses more information from the hidden state compared to the main information flow. It is an important finding. Generalizing, it can be seen as units routing the present input through a path controlled by information from both the past and present.

Cell state distribution and activation saturation analysis shown that the units in the first layer in both LSTM and GRU models work as feature amplifiers. The cells and gates do not saturate often, so the units simply scale the input. The higher layers on the other hand tend to behave more like classifiers.

It is astonishing that the network, trained character by character, learned a model able to spell correct English words and use punctuation just by sampling. With right temperature value it was able to deliver orthographically correct words in somehow varying topics.

The context memory analysis revealed that the model does keep information about the past sequences. It remembers the quotation marks precisely, even for longer periods. The concept of quotes is simple enough to be discovered by the network during training. Furthermore, the fact that it is discovered means that the quotes are sufficiently important to be useful for the prediction. Moreover, statistical analysis allowed to identify units that keep track of the past sequences based on much more ambiguous relations between the characters.

Karpathy in his article also evaluated RNNs with shuffled Linux Kernel source files. It is a far more structured dataset with characters that influence the text heavily. He reports finding interpretable units that activate inside if statements, that turn on inside comments and quotes and a cell sensitive to the depth of an expression.

The training dynamics section showed that the frequency of a word in the dataset influences the learning pace the most. However, also distribution of the occurrences, the similarity to other words and the length of the word play important roles.

It was presented that in the beginning of the training the network preferred to continue an opened quote, but after some epochs it was choosing either newlines (long sentence) or finishing the quote with a quotation mark. Moreover, the model given a question wrapped in quotation marks learned to give an answer also inside quotation marks.

Evolution of the unit representing the position in line showed that the moment it is developed coincides with the time the network begins to properly predict `\r\n`. It indicates that this exact unit is utilized in the model for newline prediction.

Word representation analysis established that the hidden states are not based on their function in a sentence or meaning, but rather on their structure on character level. It seems correct given the fact that the model uses character-level input.

This issue was addressed in the last chapter. Unfortunately, the proposed model turned out to be unfeasible to train throughout this master project.

7.1 FUTURE WORK

Character-level models in a task of machine translation could bring amazing results, like in word-level networks. However, character-level models have a huge advantage - their vocabulary is well defined and extremely small compared to word-level models. It greatly lowers the complexity of the implementation and theoretically it could decrease memory requirements, both during training and prediction.

Attention mechanism (Bahdanau et al., 2014) is a novel, extremely interesting concept that should be researched more. It was evaluated on word-level

machine translation task, where the model learned to shift its attention during translation. In character-level language modeling, a similar mechanism could make the influence of the past stronger by allowing the network to focus more on specific parts of the past. Since it is soft alignment, perhaps the whole words or expressions could be utilized in prediction. It probably could be easily visualized and evaluated. Furthermore, such attention mechanism could work even more efficiently with the proposed char-word architecture. Prediction of the next word could be done in few stages, the hidden state would be updated based on a couple of glimpses of the past with attention drawn to different places for each glimpse. It would allow the model to utilize information that are valuable only in a given context.

Convolutional Neural Networks proved their abilities in computer vision. But the idea could also be evaluated in language modeling task. As it was presented in Krizhevsky et al., 2012, the CNNs learn very clear, interpretable feature extractors. A convolutional layer applied between input and the first recurrent layer could learn to pick up some text motifs, e.g. "tho" or "ing" and utilize this information in prediction.

The issue of training difficulty and spending excessive amounts of time on cross validation of the parameters should be addressed. All available update algorithms use learning rate in one way or another. Since the optimal learning rate depends on the specific model and dataset, the network could infer what learning rate to use during the training. The way to achieve it could be difficult. Set the learning rate as another parameter of the network? Sample different learning rate values after each iteration and minimize the loss function? Apply an extra FNN that would optimize it with regards to the weights and loss function of the original network? Separate learning rate for each weight or groups of weights? How would the gradient propagate in such a solution? Split the iteration into two stages - computing the gradients and then optimize the learning rate? The solution is probably difficult to find, but it seems worth researching.

Another interesting topic is Artificial Intelligence. The trend nowadays is to prepare as much data as possible and train a model (e.g. RNN) using it. Even though better and better results are achieved, the domain of the problem is always limited. Lately, reinforcement learning showed astonishing results. The researchers managed to create a model that learned to play Atari games by trial and error. In order to deliver a real AI that would understand humans it seems crucial to expose the model to the data human beings receive during their lives. To achieve that, an AI would need a robotic body equipped with sensors equivalent to humans senses. Such sensors would produce an incredible amount of information. A neural network fed with this kind of data would make it possible to learn dependencies that humans are exposed to. The reinforcement learning techniques could optimize the network with regards to some objectives. In case of Atari games it was maximizing the score. In case of such AI a question raises: what is the point of life? Survival? The robot needs to charge itself, avoid destruction by other beings, it requires mechanical maintenance, etc. Exploration? Procreation? The question of what should be a driving force for such a robot remains rather philosophical, as it is to humans. But probably all of the above (and even more) should be combined to mimic humans objectives. In computer vision, motivated by processes in visual cortex, Convolutional Neural Networks were developed and brought astonishing performance. Probably more tricks need to be copied from human brain in order to deliver an AI similar to humans, in both behavior and understanding.

BIBLIOGRAPHY

Bahdanau, D., K. Cho, and Y. Bengio

2014 "Neural Machine Translation by Jointly Learning to Align and Translate," *ArXiv e-prints* (Sept. 2014), arXiv: 1409.0473 [cs.CL].

Bengio, Y., P. Simard, and P. Frasconi

1994 "Learning long-term dependencies with gradient descent is difficult," *IEEE Transactions on Neural Networks*, 5, 2 (Mar. 1994), pp. 157-166, ISSN: 1045-9227, DOI: 10.1109/72.279181.

Bengio, Y., N. Boulanger-Lewandowski, and R. Pascanu

2012 "Advances in Optimizing Recurrent Networks," *ArXiv e-prints* (Dec. 2012), arXiv: 1212.0901 [cs.LG].

Berglund, M., T. Raiko, M. Honkala, L. Kärkkäinen, A. Vetek, and J. Karhunen

2015 "Bidirectional Recurrent Neural Networks as Generative Models - Reconstructing Gaps in Time Series," *ArXiv e-prints* (Apr. 2015), arXiv: 1504.01575 [cs.LG].

Bishop, Christopher M.

2006 *Pattern Recognition and Machine Learning*, Springer, ISBN: 978-0387-31073-2, http://research.microsoft.com/en-us/um/people/cmbishop/prml/.

Cho, K., B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio

2014 "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation," *ArXiv e-prints* (June 2014), arXiv: 1406.1078 [cs.CL].

Chung, J., C. Gulcehre, K. Cho, and Y. Bengio

2014 "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling," *ArXiv e-prints* (Dec. 2014), arXiv: 1412.3555.

Dieleman, Sander, Eric Battenberg, Daniel Nouri, Eben Olson, Aäron van den Oord, Colin Raffel, Jan Schlüter, and Søren Kaae Sønderby

2015 *Lasagne 0.2dev1*, http://lasagne.readthedocs.org/en/latest/.

Donahue, J., L. A. Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell

2014 "Long-term Recurrent Convolutional Networks for Visual Recognition and Description," *ArXiv e-prints* (Nov. 2014), arXiv: 1411.4389 [cs.CV].

Erhan, Dumitru, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio

2010 "Why Does Unsupervised Pre-training Help Deep Learning?" 11 (Feb. 2010), pp. 625-660.

Glorot, Xavier, Antoine Bordes, and Yoshua Bengio
2011 "Deep Sparse Rectifier Neural Networks," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS-11)*, ed. by Geoffrey J. Gordon and David B. Dunson, Journal of Machine Learning Research - Workshop and Conference Proceedings, vol. 15, pp. 315-323, http://www.jmlr.org/proceedings/papers/v15/glorot11a/glorot11a.pdf.

Goodfellow, I. J., D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio
2013 "Maxout Networks," *ArXiv e-prints* (Feb. 2013), arXiv: 1302.4389 [stat.ML].

Goodman, J.
2001 "A Bit of Progress in Language Modeling," *ArXiv e-prints* (Aug. 2001), arXiv: cs/0108005.

Graves, A.
2013 "Generating Sequences With Recurrent Neural Networks," *ArXiv e-prints* (Aug. 2013), arXiv: 1308.0850.

Graves, A., A.-R. Mohamed, and G. Hinton
2013 "Speech recognition with deep recurrent neural networks," *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 6645-6649.

Graves, Alex
2012 *Supervised sequence labelling with recurrent neural networks*, Springer, vol. 385.

Graves, Alex and Jürgen Schmidhuber
2005 "Framewise phoneme classification with bidirectional {LSTM} and other neural network architectures," *Neural Networks*, 18, 5–6, {IJCNN} 2005, pp. 602 -610, ISSN: 0893-6080, DOI: http://dx.doi.org/10.1016/j.neunet.2005.06.042, http://www.sciencedirect.com/science/article/pii/S0893608005001206.

Greff, K., R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber
2015 "LSTM: A Search Space Odyssey," *ArXiv e-prints* (Mar. 2015), arXiv: 1503.04069.

Gulcehre, C., K. Cho, R. Pascanu, and Y. Bengio
2013 "Learned-Norm Pooling for Deep Feedforward and Recurrent Neural Networks," *ArXiv e-prints* (Nov. 2013), arXiv: 1311.1780.

Hannun, A., C. Case, J. Casper, B. Catanzaro, G. Diamos, E. Elsen, R. Prenger, S. Satheesh, S. Sengupta, A. Coates, and A. Y. Ng
2014 "Deep Speech: Scaling up end-to-end speech recognition," *ArXiv e-prints* (Dec. 2014), arXiv: 1412.5567 [cs.CL].

Hinton, G. E., N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov
2012 "Improving neural networks by preventing co-adaptation of feature detectors," *ArXiv e-prints* (July 2012), arXiv: 1207.0580.

Hochreiter, S.
1991 *Untersuchungen zu dynamischen neuronalen Netzen*, PhD thesis, Institut für Informatik, Lehrstuhl Prof. Brauer, Technische Universität München.

Hochreiter, S. and J. Schmidhuber

1997 "Long short-term memory," *Neural computation*, 9, 8 (June 1997), pp. 1735-1780.

Jelinek, F., B. Merialdo, S. Roukos, and M. Strauss

1991 "A Dynamic Language Model for Speech Recognition," in *Proceedings of the Workshop on Speech and Natural Language*, HLT '91, Association for Computational Linguistics, Pacific Grove, California, pp. 293-295, DOI: 10.3115/112405.112464, http://dx.doi.org/10.3115/112405.112464.

Kaae Sønderby, Søren

2015 *Penn Tree - Lasagne example*, https://github.com/skaae/Lasagne/blob/f38227d075245c5b882cfe3d18bb6502b57e4ea6/examples/pentree.py.

Kaae Sønderby, S. and O. Winther

2014 "Protein Secondary Structure Prediction with Long Short Term Memory Networks," *ArXiv e-prints* (Dec. 2014), arXiv: 1412.7828 [q-bio.QM].

Kaae Sønderby, S., C. Kaae Sønderby, H. Nielsen, and O. Winther

2015 "Convolutional LSTM Networks for Subcellular Localization of Proteins," *ArXiv e-prints* (Mar. 2015), arXiv: 1503.01919 [q-bio.QM].

Karpathy, A. and L. Fei-Fei

2014 "Deep Visual-Semantic Alignments for Generating Image Descriptions," *ArXiv e-prints* (Dec. 2014), arXiv: 1412.2306 [cs.CV].

Karpathy, A., J. Johnson, and L. Fei-Fei

2015 "Visualizing and Understanding Recurrent Networks," *ArXiv e-prints* (June 2015), arXiv: 1506.02078 [cs.LG].

Karpathy, Andrej

2015 *The Unreasonable Effectiveness of Recurrent Neural Networks*, Blog, http://karpathy.github.io/2015/05/21/rnn-effectiveness/.

Krizhevsky, A., I. Sutskever, and Geoffrey E. Hinton

2012 "ImageNet Classification with Deep Convolutional Neural Networks," pp. 1097-1105, http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

Mikolov, T., I. Sutskever, K. Chen, G. Corrado, and J. Dean

2013 "Distributed Representations of Words and Phrases and their Compositionality," *ArXiv e-prints* (Oct. 2013), arXiv: 1310.4546 [cs.CL].

Mikolov, Tomáš

2012 *Statistical Language Models Based on Neural Networks*, PhD thesis, Brno, CZ, p. 129, http://www.fit.vutbr.cz/research/view_pub.php?id=10158.

Mitchell, Thomas M.

1997 *Machine Learning*, 1st ed., McGraw-Hill, Inc., New York, NY, USA, ISBN: 0070428077, 9780070428072.

Pascanu, R., C. Gulcehre, K. Cho, and Y. Bengio
2013 "How to Construct Deep Recurrent Neural Networks," *ArXiv e-prints* (Dec. 2013), arXiv: 1312.6026.

PlayStation 4 Safety Guide 2013 (ed.), Sony Computer Entertainment Inc., China.

Rumelhart, D. E., G. E. Hinton, and R. J. Williams
1988 "Neurocomputing: Foundations of Research," in ed. by James A. Anderson and Edward Rosenfeld, MIT Press, Cambridge, MA, USA, chap. Learning Internal Representations by Error Propagation, pp. 673-695, ISBN: 0-262-01097-6, http://psych.stanford.edu/~jlm/papers/PDP/Volume%201/Chap8_PDP86.pdf.

Schmidhuber, J.
2014 "Deep Learning in Neural Networks: An Overview," *ArXiv e-prints* (Apr. 2014), arXiv: 1404.7828.

Schuster, Mike and Kuldip K. Paliwal
1997 "Bidirectional recurrent neural networks," *IEEE Transactions on Signal Processing*, 45, 11, pp. 2673-2681, http://www.di.ufpe.br/~fnj/RNA/bibliografia/BRNN.pdf.

Stallkamp, J., M. Schlipsing, J. Salmen, and C. Igel
2012 "Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition," *Neural Networks*, ISSN: 0893-6080, DOI: 10.1016/j.neunet.2012.02.016, http://www.sciencedirect.com/science/article/pii/S0893608012000457.

Sutskever, I., O. Vinyals, and Q. V. Le
2014 "Sequence to Sequence Learning with Neural Networks," *ArXiv e-prints* (Sept. 2014), arXiv: 1409.3215 [cs.CL].

Tieleman, Tijmen and Geoffrey Hinton
2012 "Lecture 6.5 - rmsprop: Divide the gradient by a running average of its recent magnitude," *COURSERA: Neural Networks for Machine Learning*, http://www.youtube.com/watch?v=O3sxAc4hxZU.

Top 500 supercomputing sites n.d. , http://top500.org/, Accessed: 2015-07-09.

Vinyals, O., A. Toshev, S. Bengio, and D. Erhan
2014 "Show and Tell: A Neural Image Caption Generator," *ArXiv e-prints* (Nov. 2014), arXiv: 1411.4555 [cs.CV].

Zaremba, W. and I. Sutskever
2014 "Learning to Execute," *ArXiv e-prints* (Oct. 2014), arXiv: 1410.4615.

Zaremba, W., I. Sutskever, and O. Vinyals
2014 "Recurrent Neural Network Regularization," *ArXiv e-prints* (Sept. 2014), arXiv: 1409.2329.

A ANSWER WITH QUOTES: EVOLUTION OF PREDICTIONS

Predictions generated by the 3LSTM512 model after each epoch of training after inputting *"How are you?"\r\n* (Section 5.5). There is \r\n in the beginning of each prediction, omitted for clarity.

Epoch 1:

"I was the counters of the count of the

Epoch 2:

"Why do you was to the soldiers and the soldiers and the soldiers and the same and the same and the same and the same and the same and the same and the same and the same and the same and the same and the same and the same and the same and the same and the same and the same and the same and the same and the same

Epoch 3:

"I am a long that the count was not to be a soldiers and the same man who had been to be to be the same that the count was not to be a soldiers and the same that the count was not to be a soldiers and the same that the count was not to be a soldiers and the same that the count was a strange of the

Epoch 4:

"Well, I was a long the same that is not the same that is not the same that is not the same that is not the same that is not the same that is not the same that were the same the same the same the same the same the same the same the same the same the same the same the same the same the same the same

Epoch 5:

"What a soldiers were the same thing is a long thing in the same thing in the same thing in the same thing in the same thing in the same thing in the same thing in the same thing in the count with a strange and the same thing in the count with a strange and the same thing were all the same thing of

Epoch 6:

"I am a strange and the same thing is not to say that I was a long time to the countess.\r\n\r\n"

Epoch 7:

"What a man who was a long time to say that they were all the same time to the countess."

Epoch 8:

"What do you think the countess were the same time to be a man who was a long time. I am always seemed to be a man who was\r\n all the same thing is not the same thing is not the same thing is not the same thing is not the same thing is not

the same thing is not the same thing is not the same thing is

Epoch 9:

"What does it is not to be a man who was a man who was all the same time to the\r\n country. I am so much a man who was all the same time to the country and the same time the\r\n countess was a strange and streets and the same time the countess was already to be a stranger and the\r\n commander of the count

Epoch 10:

"What do you think they were all the same thing is not a man who was\r\n all the same thing is not to be a man who was all the same thing\r\n and the country was a long time and the count was already to be a strange\r\n which he was already to be a strange and the same time the same time the\r\n count was alre

Epoch 11:

"What a sound of the countess will be a man who was all the same thing is\r\n the countess, and the countess was a stranger and the same time to see him\r\n and the same time to say that the count was a strange and the same time to\r\n the countess and the same time to him and the same time to him and the

Epoch 12:

"I am a strange man who was a stranger and the same time the same\r\n thing is not the same time to be a stranger and the same time the\r\n countess was a stranger and the same time the count was a stranger\r\n and the same time the count was a stranger and the same time the\r\n countess was a strange commande

Epoch 13:

"What a man who was a stranger and the same time the count was a stranger\r\n and the same time the count was a stranger and the same time the same time\r\n the count was a stranger and the same time he had been sent to him and the\r\n same time he had been sent to him and the same time he had been the same

Epoch 14:

"What a princess will be a man who was all the same time to be\r\n all the same time to say that I was all the same time to be a man who\r\n was already to be a stranger and the same time the count was already\r\n the same time the count was saying to him and the same time the\r\n count was saying to him and t

Epoch 15:

"What a strange man who was all the same time to see the country."

Epoch 16:

"What a strange man who was a stranger and the same time to say that the\r\n countess was a stranger and strength of the same time to say that he was\r\n already to be a stranger and the same time the count was already to be a strange\r\n proposal and the same time the count was still the same time to be

Epoch 17:

"What is the same time?"

Epoch 18:

"What a strange man who was a man who was all the same time to be a man who\r\n was all the same time to say that the count was a stranger and the same

time\r\n that he was saying to him and the same time he had seen him and the same\r\n time with a smile and strange street was still still still still still

Epoch 19:
"What do you think of your honor?"

Epoch 20:
"What a strange man who was a stranger and the same time they were\r\n so much as the count was already and so strange and so strange and the\r\n same time the count was saying and said that he was already and so\r\n that he was already and something to him and the same time he had\r\n seen him and said that h

Epoch 21:
"I am so much as the countess will be a stranger and the same time to\r\n see you to be a commander-in-chief. I will tell you the countess, and\r\n the countess was a man who was already and the same time the countess\r\n was a strange commander-in-chief's face and the same time he had been\r\n advanced and th

Epoch 22:
"What a strange forest was a man who was all the same time, and the\r\n countess will be a strange for the commander-in-chief. I am not a man who\r\n was a strange forehead and the same time the countess was a strange\r\n particular position and the same time the countess was already\r\n seen and the same time

Epoch 23:
"What a strange strength of men who have been so much as they were saying?"

Epoch 24:
"What a strange soul I am so much as you are all the same time."

Epoch 25:
"I am so much as the count will be a man who was all the same time. I am\r\n all the same time to see him and the same time he was saying. "

Epoch 26:
"What a strange son will be a single state of men who was a stranger\r\n that we are all the same time and the count will be a soldier who was all\r\n the same time and the same time he was saying to him and he was already\r\n so strong to him and he was saying to him and he was saying to him and\r\n said that

Epoch 27:
"I am not a stranger to the country. I am not a commander-in-chief. I am\r\n all the same to the country and the countess will be a stranger and\r\n the same time to see the countess and the same time the countess\r\n went out of the room and said that he was saying to him and he was\r\n saying to him and he w

Epoch 28:
"What a stranger was to be a man who was all the same!"

Epoch 29:
"What a strange son, you see, I will say to you, I will see the same time\r\n to be a man who was a man who was a man who was a man who was already\r\n able to say that the count was a stranger and the same time to be a soldier who\r\n

had been so strange and the same time to be a soldier who was already an

Epoch 30:
"I am so strong to him and the count was already and we have\r\n seen him and the countess was a stranger and the same time the\r\n countess was so strange to him and he was saying to him and he\r\n was saying to him and he was saying.\r\n\r\n"

Epoch 31:
"What a strange soul I want to say to him?"

Epoch 32:
"What a strange state of men who were the same!"

Epoch 33:
"What are you going?"

Epoch 34:
"What a strange son was a man who was already to be a soldier who was\r\n saying to him and that is what I should like to say that the countess was\r\n saying to him and had he not been so strongly and so that he was already been\r\n addressed to him and had he not been so strongly and so that he was alread

Epoch 35:
"What do you want?"

Epoch 36:
"What are you going?"

Epoch 37:
said Prince Andrew.\r\n\r\n"What are you going?"

Epoch 38:
"Why do you think of the commander-in-chief?"

Epoch 39:
"Why do you think of the same time?"

Epoch 40:
"What are you thinking? What do you think? What do you think? What does\r\n the world?"

Epoch 41:
"What a stranger should be a single strength?"

Epoch 42:
"Why do you want to see the count?"

Epoch 43:
"What a stranger should be a stranger and the same time?"

Epoch 44:

"What a strange strength of the same time I am so strange to see you."

Epoch 45:
"Why do you say to you?"

Epoch 46:
"Why do you think of the commander-in-chief?"

Epoch 47:
"I am so strong to see you to see you to see you to see the count's\r\n and the same time to see him and the count will be a soldier who has\r\n been so strange to see him and the count will be so strong to her. What\r\n does it mean?"

Epoch 48:
"Why don't you say that?"

Epoch 49:
"What a strange face of the count was a man who has been said to\r\n him?"

Epoch 50:
"Why do you say that?"

B | 3LSTM64: TRAINING CODE

The listing below presents the code used for training the 3LSTM64 model. The file was built upon *pentree.py* example (Kaae Sønderby, 2015).

```
from __future__ import print_function, division
import numpy as np
import theano
import theano.tensor as T
import os
import time
import lasagne
import scipy.io as sio
import pickle
import logging
import argparse

# PARSE SETTINGS
parser = argparse.ArgumentParser()
parser.add_argument("-BATCH_SIZE", type=int, default=100)
parser.add_argument("-DATA", type=str, default='warpeace_input.txt')
parser.add_argument("-MODEL_SEQ_LEN", type=int, default=100)
parser.add_argument("-TOL", type=float, default=1e-6)
parser.add_argument('-REC_NUM_UNITS', nargs='+', type=int,
                    default=[37, 37, 37])
parser.add_argument('-TRAIN_FRACTION', type=float, default=0.8)
parser.add_argument('-VAL_FRACTION', type=float, default=0.1)
parser.add_argument("-DROPOUT_FRACTION", type=float, default=0.08)
parser.add_argument("-LEARNING_RATE", type=float, default=2e-3)
parser.add_argument("-DECAY", type=float, default=0.95)
parser.add_argument("-NO_DECAY_EPOCHS", type=int, default=10)
parser.add_argument("-MAX_GRAD", type=float, default=5)
parser.add_argument("-NUM_EPOCHS", type=int, default=50)
parser.add_argument("-SEED", type=int, default=1234)
parser.add_argument("-INIT_RANGE", type=float, default=0.08)

args = parser.parse_args()

BATCH_SIZE = args.BATCH_SIZE
DATA = args.DATA
MODEL_SEQ_LEN = args.MODEL_SEQ_LEN
TOL = args.TOL
REC_NUM_UNITS = args.REC_NUM_UNITS
if isinstance(REC_NUM_UNITS, (int, long)):
    REC_NUM_UNITS = [REC_NUM_UNITS]
assert len(REC_NUM_UNITS) == 3
SPLIT_SIZES = [args.TRAIN_FRACTION, args.VAL_FRACTION]
DROPOUT_FRACTION = args.DROPOUT_FRACTION
LEARNING_RATE = args.LEARNING_RATE
DECAY = args.DECAY
NO_DECAY_EPOCHS = args.NO_DECAY_EPOCHS
MAX_GRAD = args.MAX_GRAD
NUM_EPOCHS = args.NUM_EPOCHS
```

```
np.random.seed(args.SEED)
INI = lasagne.init.Uniform(args.INIT_RANGE)

# CREATE OUTPUT FOLDER
folder_name = '%d_LSTM' % len(REC_NUM_UNITS)
for rnu in REC_NUM_UNITS:
    folder_name += '_%d' % rnu
folder_name = 'output/' + folder_name
if not os.path.exists(folder_name):
    os.makedirs(folder_name)

# INITIALIZE LOGGER
logger = logging.getLogger('')
logger.setLevel(logging.DEBUG)
fh = logging.FileHandler(folder_name + "/train_output.log", mode='w')
fh.setLevel(logging.DEBUG)
ch = logging.StreamHandler()
ch.setLevel(logging.DEBUG)
formatter = logging.Formatter('%(message)s')
ch.setFormatter(formatter)
fh.setFormatter(formatter)
logger.addHandler(ch)
logger.addHandler(fh)

# LOG PARAMETERS
logger.info('Settings')
logger.info('BATCH_SIZE      : ' + str(BATCH_SIZE))
logger.info('DATA            : ' + str(DATA))
logger.info('MODEL_SEQ_LEN   : ' + str(MODEL_SEQ_LEN))
logger.info('TOL             : ' + str(TOL))
logger.info('REC_NUM_UNITS   : ' + str(REC_NUM_UNITS))
logger.info('SPLIT_SIZES     : ' + str(SPLIT_SIZES))
logger.info('DROPOUT_FRACTION: ' + str(DROPOUT_FRACTION))
logger.info('LEARNING_RATE   : ' + str(LEARNING_RATE))
logger.info('DECAY           : ' + str(DECAY))
logger.info('NO_DECAY_EPOCHS : ' + str(NO_DECAY_EPOCHS))
logger.info('MAX_GRAD        : ' + str(MAX_GRAD))
logger.info('NUM_EPOCHS      : ' + str(NUM_EPOCHS))
logger.info('RNG SEED        : ' + str(args.SEED))
logger.info('INIT RANGE      : ' + str(args.INIT_RANGE))
logger.info('folder_name     : ' + str(folder_name))

# GET FLOATX TYPE
floatX_dtype = np.dtype(theano.config.floatX).type

# DEFINE DATA LOADING AND PREPROCESSING FUNCTIONS
def load_data(file_name):
    vocab_size = 0
    vocab_map = {}
    with open(file_name, 'rb') as f:
        raw_data = f.read()
        x = np.zeros(raw_data.__len__())
        for iSym, symbol in enumerate(raw_data):
            if symbol not in vocab_map:
                vocab_map[symbol] = vocab_size
                vocab_size += 1
            x[iSym] = vocab_map[symbol]
    print("Loaded %i symbols from %s" % (iSym, file_name))
```

```
    return x.astype('int32'), vocab_map, vocab_size


def reorder(x_in, batch_size, model_seq_len, vocab_size):
    if x_in.shape[0] % (batch_size*model_seq_len) == 0:
        x_in = x_in[:-1]

    x_resize = (x_in.shape[0] // (batch_size*model_seq_len)) * \
               model_seq_len*batch_size
    n_samples = x_resize // model_seq_len
    n_batches = n_samples // batch_size

    targets = x_in[1:x_resize+1]
    x_out_ = x_in[:x_resize]

    x_out = np.zeros((x_out_.shape[0], vocab_size)).astype('int32')
    for iSym, symbol in enumerate(x_out_):
        x_out[iSym, symbol] = 1

    x_out = x_out.reshape((n_samples, model_seq_len, vocab_size))
    targets = targets.reshape((n_samples, model_seq_len))

    out = np.zeros(n_samples, dtype=int)
    for i in range(n_batches):
        val = range(i, n_batches*batch_size+i, n_batches)
        out[i*batch_size:(i+1)*batch_size] = val

    x_out = x_out[out]
    targets = targets[out]

    return x_out.astype(floatX_dtype), targets.astype('int32')


def get_data(model_seq_len, batch_size, split_sizes, data_file):
    x, vocab_map, vocab_size = load_data(os.path.join('data', data_file))

    n_train = np.floor(split_sizes[0] * x.shape[0])
    n_val = np.floor(split_sizes[1] * x.shape[0])
    x_train = x[:n_train]
    x_val = x[n_train: n_train+n_val]
    x_test = x[n_train+n_val:]

    x_train, y_train = \
        reorder(x_train, batch_size, model_seq_len, vocab_size)
    x_val, y_val = \
        reorder(x_val, batch_size, model_seq_len, vocab_size)
    x_test, y_test = \
        reorder(x_test, batch_size, model_seq_len, vocab_size)

    return x_train, y_train, x_val, y_val, x_test, y_test, vocab_map,
        vocab_size

# LOAD AND PREPROCESS DATA
t0 = time.time()
x_train, y_train, x_val, y_val, x_test, y_test, vocab_map, vocab_size = \
    get_data(MODEL_SEQ_LEN, BATCH_SIZE, SPLIT_SIZES, DATA)
logger.info('Loading and reordering data took  %.2fs' % (time.time()-t0))
```

```
# LOG DATA PROPERTIES
logger.info("-" * 80)
logger.info("Vocabulary size: " + str(vocab_size))
logger.info("Data shapes")
logger.info("Train data     : " + str(x_train.shape))
logger.info("Validation data: " + str(x_val.shape))
logger.info("Test data      : " + str(x_test.shape))
logger.info("-" * 80)

# DEFINE THEANO SYMBOLIC VARIABLES
sym_x = T.tensor3(dtype=theano.config.floatX)
sym_y = T.imatrix()
cell1_init_sym = T.matrix(dtype=theano.config.floatX)
hid1_init_sym = T.matrix(dtype=theano.config.floatX)
cell2_init_sym = T.matrix(dtype=theano.config.floatX)
hid2_init_sym = T.matrix(dtype=theano.config.floatX)
cell3_init_sym = T.matrix(dtype=theano.config.floatX)
hid3_init_sym = T.matrix(dtype=theano.config.floatX)

# BUILD THE MODEL

# INPUT
l_inp = lasagne.layers.InputLayer(
    shape=(BATCH_SIZE, MODEL_SEQ_LEN, vocab_size),
    input_var=sym_x)

# DROPOUT
l_in_reshape = lasagne.layers.ReshapeLayer(
    l_inp,
    (BATCH_SIZE*MODEL_SEQ_LEN, vocab_size))

l_drp0 = lasagne.layers.DropoutLayer(
    l_in_reshape,
    p=DROPOUT_FRACTION)

l_drp0_reshape = lasagne.layers.ReshapeLayer(
    l_drp0,
    (BATCH_SIZE, MODEL_SEQ_LEN, vocab_size))

# FIRST LSTM
l_rec1 = lasagne.layers.LSTMLayer(
    l_drp0_reshape,
    num_units=REC_NUM_UNITS[0],
    peepholes=False,
    ingate=lasagne.layers.Gate(W_in=INI, W_hid=INI, W_cell=INI),
    forgetgate=lasagne.layers.Gate(W_in=INI, W_hid=INI, W_cell=INI),
    outgate=lasagne.layers.Gate(W_in=INI, W_hid=INI, W_cell=INI),
    cell=lasagne.layers.Gate(W_in=INI, W_hid=INI, W_cell=None,
                             nonlinearity=lasagne.nonlinearities.tanh),
    learn_init=False,
    cell_init=cell1_init_sym,
    hid_init=hid1_init_sym,
    precompute_input=True,
    grad_clipping=MAX_GRAD)

# DROPOUT
l_drp1 = lasagne.layers.DropoutLayer(
    l_rec1,
```

```
    p=DROPOUT_FRACTION)

# SECOND LSTM
l_rec2 = lasagne.layers.LSTMLayer(
    l_drp1,
    num_units=REC_NUM_UNITS[1],
    peepholes=False,
    ingate=lasagne.layers.Gate(W_in=INI, W_hid=INI, W_cell=INI),
    forgetgate=lasagne.layers.Gate(W_in=INI, W_hid=INI, W_cell=INI),
    outgate=lasagne.layers.Gate(W_in=INI, W_hid=INI, W_cell=INI),
    cell=lasagne.layers.Gate(W_in=INI, W_hid=INI, W_cell=None,
                             nonlinearity=lasagne.nonlinearities.tanh),
    learn_init=False,
    cell_init=cell2_init_sym,
    hid_init=hid2_init_sym,
    precompute_input=True,
    grad_clipping=MAX_GRAD)

# DROPOUT
l_drp2 = lasagne.layers.DropoutLayer(
    l_rec2,
    p=DROPOUT_FRACTION)

# THIRD LSTM
l_rec3 = lasagne.layers.LSTMLayer(
    l_drp2,
    num_units=REC_NUM_UNITS[2],
    peepholes=False,
    ingate=lasagne.layers.Gate(W_in=INI, W_hid=INI, W_cell=INI),
    forgetgate=lasagne.layers.Gate(W_in=INI, W_hid=INI, W_cell=INI),
    outgate=lasagne.layers.Gate(W_in=INI, W_hid=INI, W_cell=INI),
    cell=lasagne.layers.Gate(W_in=INI, W_hid=INI, W_cell=None,
                             nonlinearity=lasagne.nonlinearities.tanh),
    learn_init=False,
    cell_init=cell3_init_sym,
    hid_init=hid3_init_sym,
    precompute_input=True,
    grad_clipping=MAX_GRAD)

# DROPOUT
l_drp3 = lasagne.layers.DropoutLayer(
    l_rec3,
    p=DROPOUT_FRACTION)

l_shp = lasagne.layers.ReshapeLayer(
    l_drp3,
    (BATCH_SIZE*MODEL_SEQ_LEN, REC_NUM_UNITS[2]))

# SOFTMAX OUTPUT
l_out = lasagne.layers.DenseLayer(
    l_shp,
    num_units=vocab_size,
    nonlinearity=lasagne.nonlinearities.softmax)

# DEFINE LOSS FUNCTION
def cross_ent(net_output, target):
    predictions = \
        T.reshape(net_output, (BATCH_SIZE*MODEL_SEQ_LEN, vocab_size))
```

```
    predictions += TOL
    target = T.flatten(target)
    cost_ce = T.nnet.categorical_crossentropy(predictions, target)
    return T.sum(cost_ce)

# DEFINE THEANO VARIABLES FOR TRAINING AND EVALUATION
train_out = lasagne.layers.get_output(l_out, deterministic=False)
hidden_states_train = [l_rec1.cell_out, l_rec1.hid_out,
                       l_rec2.cell_out, l_rec2.hid_out,
                       l_rec3.cell_out, l_rec3.hid_out]
cost_train = cross_ent(train_out, sym_y)

eval_out = lasagne.layers.get_output(l_out, deterministic=True)
hidden_states_eval = [l_rec1.cell_out, l_rec1.hid_out,
                      l_rec2.cell_out, l_rec2.hid_out,
                      l_rec3.cell_out, l_rec3.hid_out]
cost_eval = cross_ent(eval_out, sym_y)

# GET ALL PARAMETERS OF THE NETWORK
all_params = lasagne.layers.get_all_params(l_out, trainable=True)

# LOG THE PARAMETERS
logger.info("-" * 80)
total_params = sum([p.get_value().size for p in all_params])
logger.info("#NETWORK params: " + str(total_params))
logger.info("Parameters:")
logger.info([{a.name, a.get_value().shape} for a in all_params])
logger.info("-" * 80)

# DEFINE THEANO VARIABLE FOR GRADIENT
all_grads = T.grad(cost_train, all_params)

# DEFINE UPDATE RULES
sh_lr = theano.shared(lasagne.utils.floatX(LEARNING_RATE))
updates = lasagne.updates.rmsprop(all_grads, all_params,
                                  learning_rate=sh_lr)

# DEFINE EVALUATION AND TRAIN FUNCTIONS
fun_inp = [sym_x, sym_y,
           cell1_init_sym, hid1_init_sym,
           cell2_init_sym, hid2_init_sym,
           cell3_init_sym, hid3_init_sym]

logger.info("compiling f_eval...")
f_eval = theano.function(fun_inp,
                         [cost_eval,
                          hidden_states_eval[0][:, -1],
                          hidden_states_eval[1][:, -1],
                          hidden_states_eval[2][:, -1],
                          hidden_states_eval[3][:, -1],
                          hidden_states_eval[4][:, -1],
                          hidden_states_eval[5][:, -1]],
                         allow_input_downcast=True)

logger.info("compiling f_train...")
f_train = theano.function(fun_inp,
                          [cost_train,
                           hidden_states_train[0][:, -1],
```

```
                          hidden_states_train[1][:, -1],
                          hidden_states_train[2][:, -1],
                          hidden_states_train[3][:, -1],
                          hidden_states_train[4][:, -1],
                          hidden_states_train[5][:, -1]],
                         updates=updates,
                         allow_input_downcast=True)

# DEFINE HELPER FUNCTION FOR DATASET EVALUATION
def calc_cross_entropy(x, y):
    n_batches = x.shape[0] // BATCH_SIZE
    l_cost = []
    cell1, hid1 = [np.zeros((BATCH_SIZE, REC_NUM_UNITS[0]),
                            dtype=floatX_dtype) for _ in range(2)]
    cell2, hid2 = [np.zeros((BATCH_SIZE, REC_NUM_UNITS[1]),
                            dtype=floatX_dtype) for _ in range(2)]
    cell3, hid3 = [np.zeros((BATCH_SIZE, REC_NUM_UNITS[2]),
                            dtype=floatX_dtype) for _ in range(2)]
    for i in range(n_batches):
        x_batch = x[i*BATCH_SIZE:(i+1)*BATCH_SIZE]
        y_batch = y[i*BATCH_SIZE:(i+1)*BATCH_SIZE]
        cost, cell1, hid1, cell2, hid2, cell3, hid3 = f_eval(
            x_batch, y_batch, cell1, hid1, cell2, hid2, cell3, hid3)
        l_cost.append(cost)

    ce = np.mean(l_cost) / BATCH_SIZE / MODEL_SEQ_LEN
    return ce


# MAIN TRAINING LOOP
logger.info("Begin training...")
ce_train = []
ce_valid = []
ce_test = []
n_batches_train = x_train.shape[0] // BATCH_SIZE

for epoch in range(NUM_EPOCHS):
    l_cost, batch_time = [], time.time()

    # INITIALIZE HIDDEN STATE WITH ZEROS
    cell1, hid1 = [np.zeros((BATCH_SIZE, REC_NUM_UNITS[0]),
                            dtype=floatX_dtype) for _ in range(2)]
    cell2, hid2 = [np.zeros((BATCH_SIZE, REC_NUM_UNITS[1]),
                            dtype=floatX_dtype) for _ in range(2)]
    cell3, hid3 = [np.zeros((BATCH_SIZE, REC_NUM_UNITS[2]),
                            dtype=floatX_dtype) for _ in range(2)]

    # TRAIN ALL THE BATCHES
    for i in range(n_batches_train):
        # FETCH DATA
        x_batch = x_train[i*BATCH_SIZE:(i+1)*BATCH_SIZE]
        y_batch = y_train[i*BATCH_SIZE:(i+1)*BATCH_SIZE]

        # TRAIN
        cost, cell1, hid1, cell2, hid2, cell3, hid3 = f_train(
            x_batch, y_batch, cell1, hid1, cell2, hid2, cell3, hid3)
        l_cost.append(cost)
```

```
    # APPLY LEARNING RATE DECAY
    if epoch > (NO_DECAY_EPOCHS - 1):
        current_lr = sh_lr.get_value()
        sh_lr.set_value(lasagne.utils.floatX(current_lr * float(DECAY)))

    # EVALUATE AND LOG
    elapsed = time.time() - batch_time
    chars_per_second = \
        float(BATCH_SIZE*MODEL_SEQ_LEN*n_batches_train) / elapsed
    crossent_valid = calc_cross_entropy(x_val, y_val)
    crossent_train = np.mean(l_cost) / BATCH_SIZE / MODEL_SEQ_LEN
    crossent_test = calc_cross_entropy(x_test, y_test)
    logger.info("*------------ Epoch: " + str(epoch))
    logger.info("Cross entropy train: " + str(crossent_train))
    logger.info("Cross entropy valid: " + str(crossent_valid))
    logger.info("Cross entropy test : " + str(crossent_test))
    logger.info("Chars per second   : " + str(chars_per_second))
    logger.info("Time elapsed       : " + str(int(elapsed)) + ' s')
    logger.info("ETA                : " +
                str(int(elapsed*(NUM_EPOCHS-epoch-1)/60)) + ' min')

    ce_train.append(crossent_train)
    ce_valid.append(crossent_valid)
    ce_test.append(crossent_test)

    # STORE CROSS ENTROPY
    sio.savemat(folder_name + '/ce.mat', {'ce_train': ce_train,
                                          'ce_valid': ce_valid,
                                          'ce_test': ce_test})

    # STORE PARAMETERS
    pickle.dump(all_params,
                open(folder_name + '/params_' + str(epoch) + '.p', "wb"))

print("done.")
```

C CONTEXT MEMORY: CELL STATES

Figure 37, Figure 38, Figure 39, Figure 40, Figure 41, Figure 42 and Figure 43 below present 15 words with their position on y axis controlled by the cell state of one particular unit. Each unique word is spread evenly on the x axis (see Subsection 5.4.1).

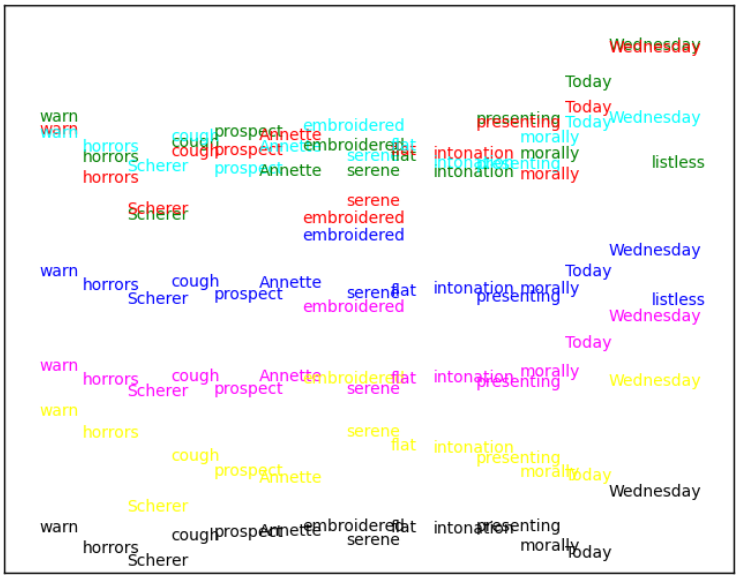


Figure 37: Cell state (y axis) of the unit that turns on inside quotes after inputting sentences with appended words.

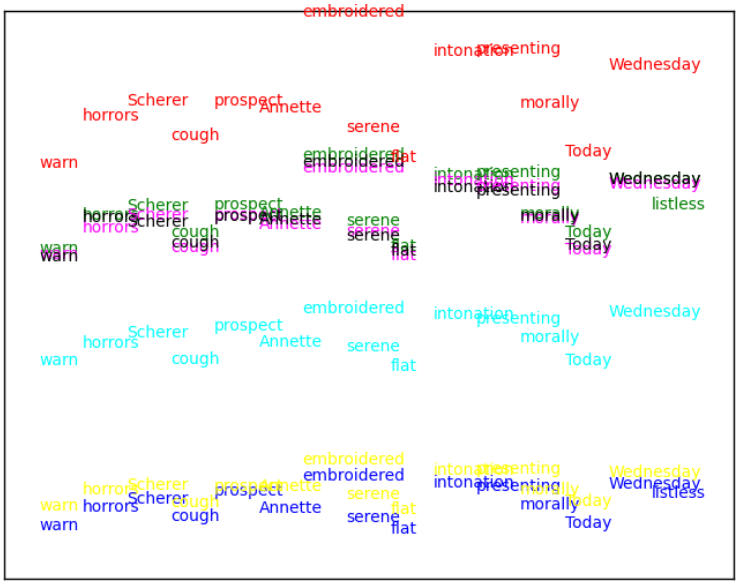


Figure 38: Cell state (y axis) of the unit representing the position in line after inputting sentences with appended words.

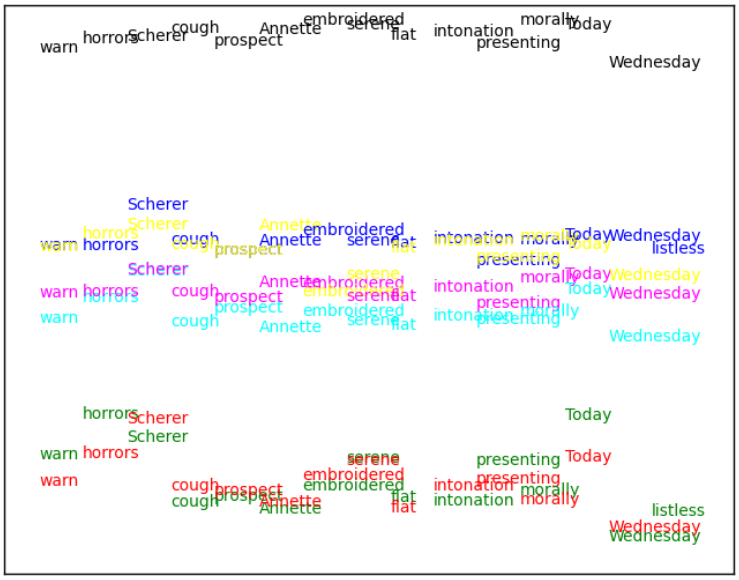


Figure 39: Cell state (y axis) of the unit fuzzily turning inside quotes after inputting sentences with appended words.

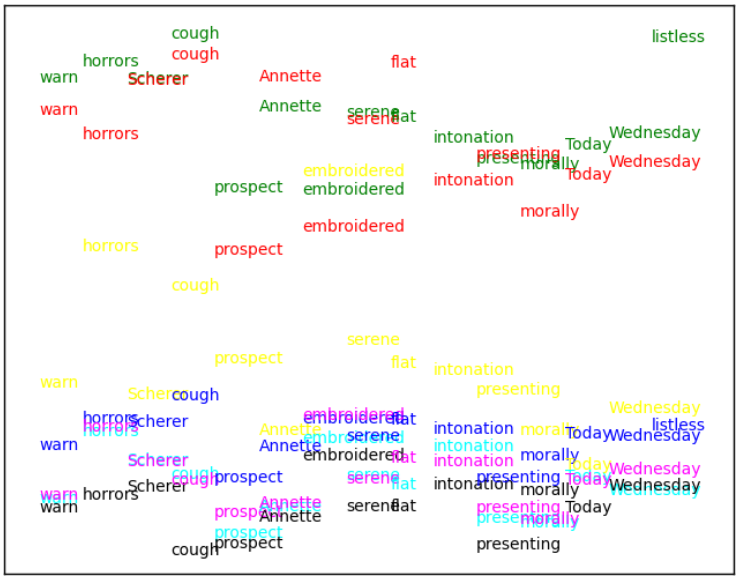


Figure 40: Cell state (y axis) of the unit fuzzily turning inside quotes (another one) after inputting sentences with appended words.

Figure 44, Figure 45 and Figure 46 depict cell states as a background color of a part of the validation set, for the last three units, in the same order.

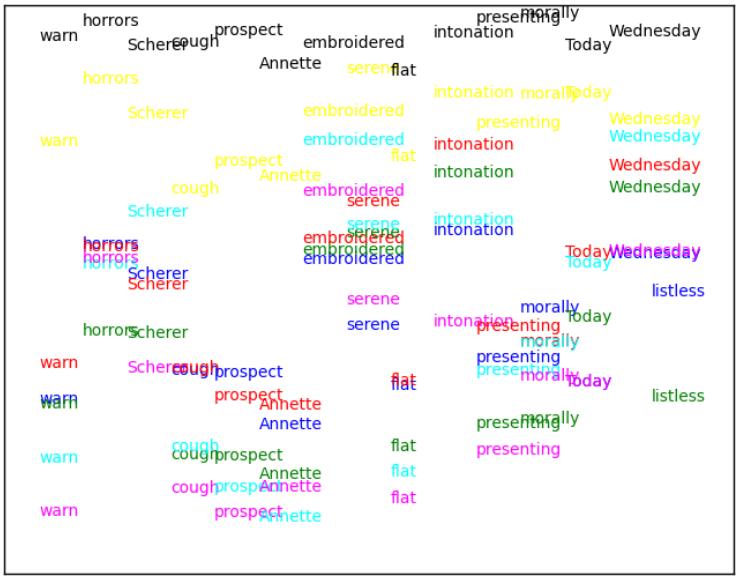


Figure 41: Cell state (y axis) of the unit with difficult interpretation no. 1 after inputting sentences with appended words. It activates on some chosen spaces, colons, periods, commas, exclamation marks a a few more characters.

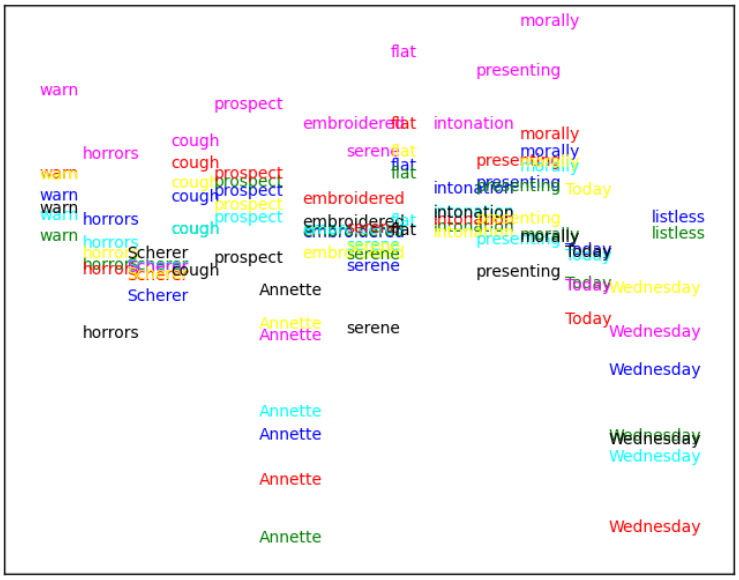


Figure 42: Cell state (y axis) of the unit with difficult interpretation no. 2 after inputting sentences with appended words. It activates on "Princess Mary", "Yaroslavl", "excellency", on french translation and more.

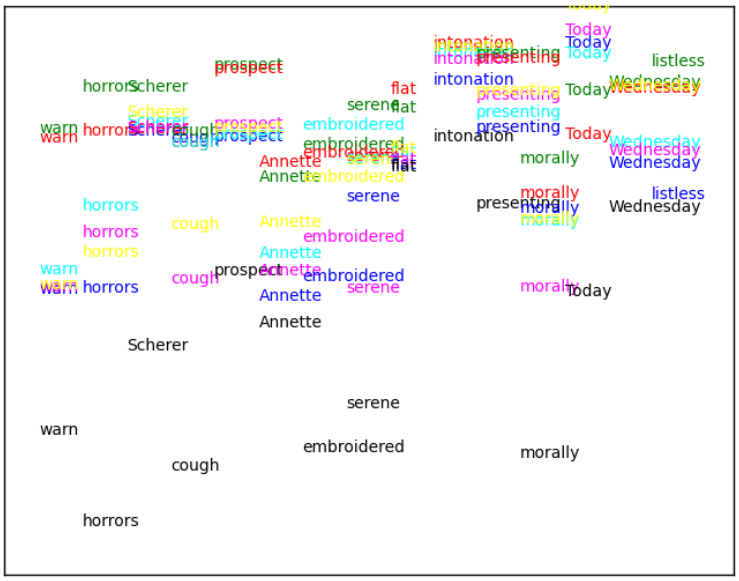


Figure 43: Cell state (y axis) of the unit with difficult interpretation no. 3 after inputting sentences with appended words. It activates on "Princess Mary", "Yaroslavl", "Prince Andrew" and more.

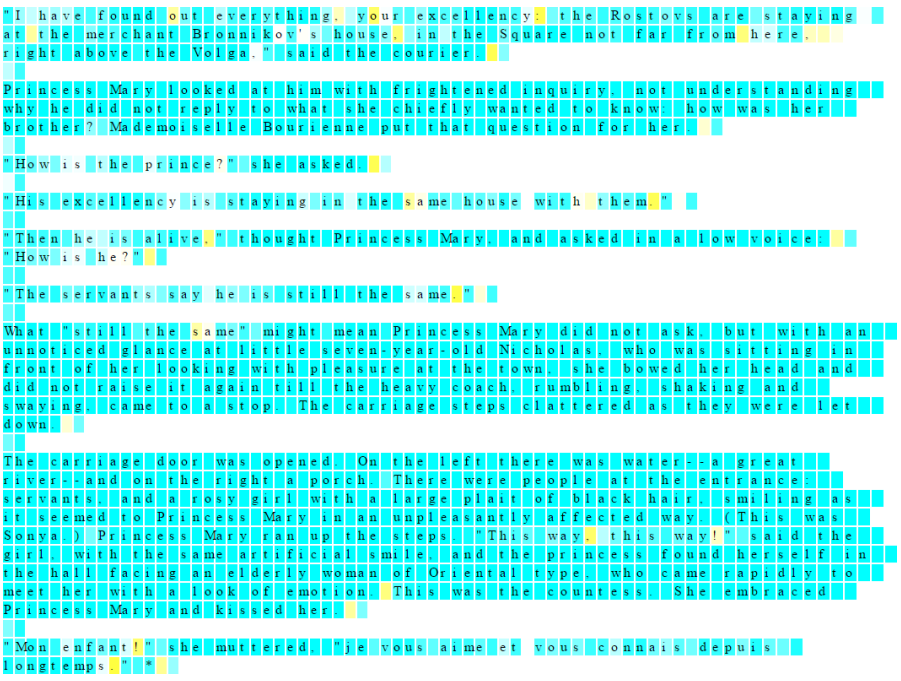


Figure 44: Cell state activations of the unit that is difficult to interpret no. 1. It activates on some chosen spaces, colons, periods, commas, exclamation marks a a few more characters. Color represents the value of the activation, with yellow denoting 1, white 0 and blue -1. Presenetd on a part of the validation set.

Figure 45: Cell state activations of the unit that is difficult to interpret no. 2. It activates on "Princess Mary", "Yaroslavl", "excellency", on french translation and more. Color represents the value of the activation, with yellow denoting 1, white 0 and blue -1. Presenetd on a part of the validation set.

Figure 46: Cell state activations of the unit that is difficult to interpret no. 3. It activates on "Princess Mary", "Yaroslavl", "Prince Andrew" and more. Color represents the value of the activation, with yellow denoting 1, white 0 and blue -1. Presenetd on a part of the validation set.

D NETWORK SIZES

Table 6: Exact sizes of the networks used in the experiments. The values represent the number of units per layer. All models in a given row have approximately the same number of parameters. Adapted from Karpathy et al., 2015.

	LSTM			GRU		
Layers	1	2	3	1	2	3
Size						
64	64	45	37	77	53	44
128	128	89	68	151	98	79
256	256	159	126	300	185	146
512	512	308	241	596	357	280