Rheinisch-Westfälische Technische Hochschule Aachen
Lehrstuhl für Informatik 6
Prof. Dr.-Ing. Hermann Ney

Selected Topics in Human Language Technology and Pattern Recognition (WS 2015)

Sequence Generation with Recurrent Neural Networks

Arne Nix

Matriculation number 331423

19/01/2016

Supervisor: Patrick Doetsch

Contents

List of Figures

1 Introduction

Machine learning techniques and algorithms have become a major building block of modern computer systems. Especially artificial neural networks have experienced a renaissance in the last years. They have proven to outperform traditional models in various disciplines. Among these are classification, image recognition, speech recognition, machine translation and many more.

Some of these disciplines are different in the sense that they do not have a fixed size in input or output. Such problems that have a vector sequence as input or output can not be solved using standard feed forward neural networks since the number of neurons in the input layer and, therefore, the length of the input is fixed for every model. A solution is possible using recurrent neural networks (RNNs) which introduce cyclic connections in the hidden layers and enable sequence processing. But standard recurrent neural networks do not provide the performance on long-term dependencies they promised when originally introduced. Therefore, it is necessary to use special units like Long Short Term Memory or Gated Recurrent Unit to increase the performance of these systems.

The problem of *sequence generation* that will be the main topic of this paper is one of many tasks, for which RNN models show promising results. This problem has numerous practical applications, such as speech synthesis, handwriting generation and machine translation. For all these applications exist similar and even identical RNN models that show very exciting results. One of these models, that was originally introduced for machine translation tasks, is an RNN-encoder and decoder model [21], which tries to solve the problem of *sequence to sequence generation* as a special case of sequence generation.

Definition 1 (Sequence to Sequence Generation Problem)
Given a training set *of source sequences* $\mathbf{x} = (x(1), \dots, x(T)) \in (\mathbb{R}^n \times \dots \times \mathbb{R}^n)$ *and corresponding target sequences* $\mathbf{y} = (y(1), \dots, y(T')) \in (\mathbb{R}^n \times \dots \times \mathbb{R}^n)$ *one seeks to generate the most probable output sequence* $\mathbf{y}^*$ *for the unseen test sequence* $\mathbf{x}^*$*, i.e. to maximize* $Pr(\mathbf{y}^* | \mathbf{x}^*)$*.*

In this work, we chose conversation modelling to further explore the capabilities of generative models. Conversation modelling presents a very difficult task since the input as well as the output are sequences. Additionally, conversations mostly consist of more than one turn. It is therefore necessary to introduce long-term dependencies in order to model the response with respect to the whole context of the previous conversation. However, if models are able to successfully predict conversation utterances, this would be a great step in understanding natural language. That would imply, that the model is capable of extracting the semantic meaning of a long-range context and generate a fitting response.

Remark 1 (Notation)

1. *We use bold typeface to distinguish sequences* $\mathbf{x} = (x(1), \dots, x(T))$ *from vectors* x*.*

2. *Time steps and therefore sequence indices are introduced as a parameter* $x(t)$ *for the corresponding variable* x*.*

3. *We indicate the affiliation of a variable or parameter to a certain layer* l *by writing it in the exponent, e.g.* $x^{(l)}$ *belongs to layer* l*.*

4. *Parameters* w_{ji} *correspond to a connection from* i *to* j*.*

1.1 Outline

In this paper, we discuss sequence generation and the model proposed by Sutskever et al. in [21]. We start with an introduction of artificial neural networks (section 2), the special case of recurrent neural networks and the training methods for these structures (section 2.1). Additionally, we explain the architecture of long short term memory and gated recurrent units, which are used in most of the state of the art recurrent neural network architectures (section 2.2). Then we discuss the general problem of generating sequences (section 3) and the encoder-decoder model proposed by Sutskever et al., as a possible solution to this problem (section 3.1). In the last section, we explain the task of conversation modelling (section 4) and debate different approaches (section 4.1) as well as different evaluation techniques (section 4.2).

2 Recurrent Neural Networks (RNNs)

The idea of neural networks has been first introduced in 1943 by McCulloch and Pitts [15] and became popular in 1962 when Rosenblatt proved that simple neural networks could learn any representable function. Neural networks are inspired by the biological structure of the human brain. Simple units which effectively compute a simple function f_j with respect to the inputs x_j, a bias b_i and the corresponding weights w_{ij} are called neurons.

$$y_i = f_i(\sum_{j=1}^{J} w_{ij} \cdot x_j + b_i) \tag{1}$$

These neurons are organized in layers with weighted connections between different layers. The simplest form of an artificial neural network would be a multilayered perceptron (MLP) [18, 23, 3] which consists of an input layer, one or multiple hidden layers and an output layer, as illustrated in figure 1. An input vector x for such a neural network is presented to the neurons $i_1, ..., i_{N^{(0)}}$ of the input layer. The output y would be given by the output of neurons from the output layer after a complete forward pass through all L layers of the neural network. This means that every layer l computes its output from the output $y^{(l-1)}$ of the previous layer $l-1$ multiplied by the weight matrix $W_{\text{in}}^{(l)}$ and a bias vector $b^{(l)}$.

$$y^{(l)} = f^{(l)}(W_{\text{in}}^{(l)} y^{(l-1)} + b^{(l)}) \tag{2}$$

The activation function $f^{(l)}$, which is applied componentwise, can be a linear function, e.g. for the output layer, or a non-linear function. Popular choices for activation functions are the logistic sigmoid σ and the hyperbolic tangent $tanh$.

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{3}$$

$$tanh(x) = \frac{e^{2x}-1}{e^{2x}+1} \tag{4}$$

MLPs and variation of such feed forward neural networks are very popular and can be extremely powerful [13], but they lack certain properties which are necessary for various tasks. Feed forward neural networks depend on a fixed input size, which results from the fixed dimensionality of their input layer. They also are not aware of the context of their input, i.e. the previous or following input. This is all due to the fact that feed forward

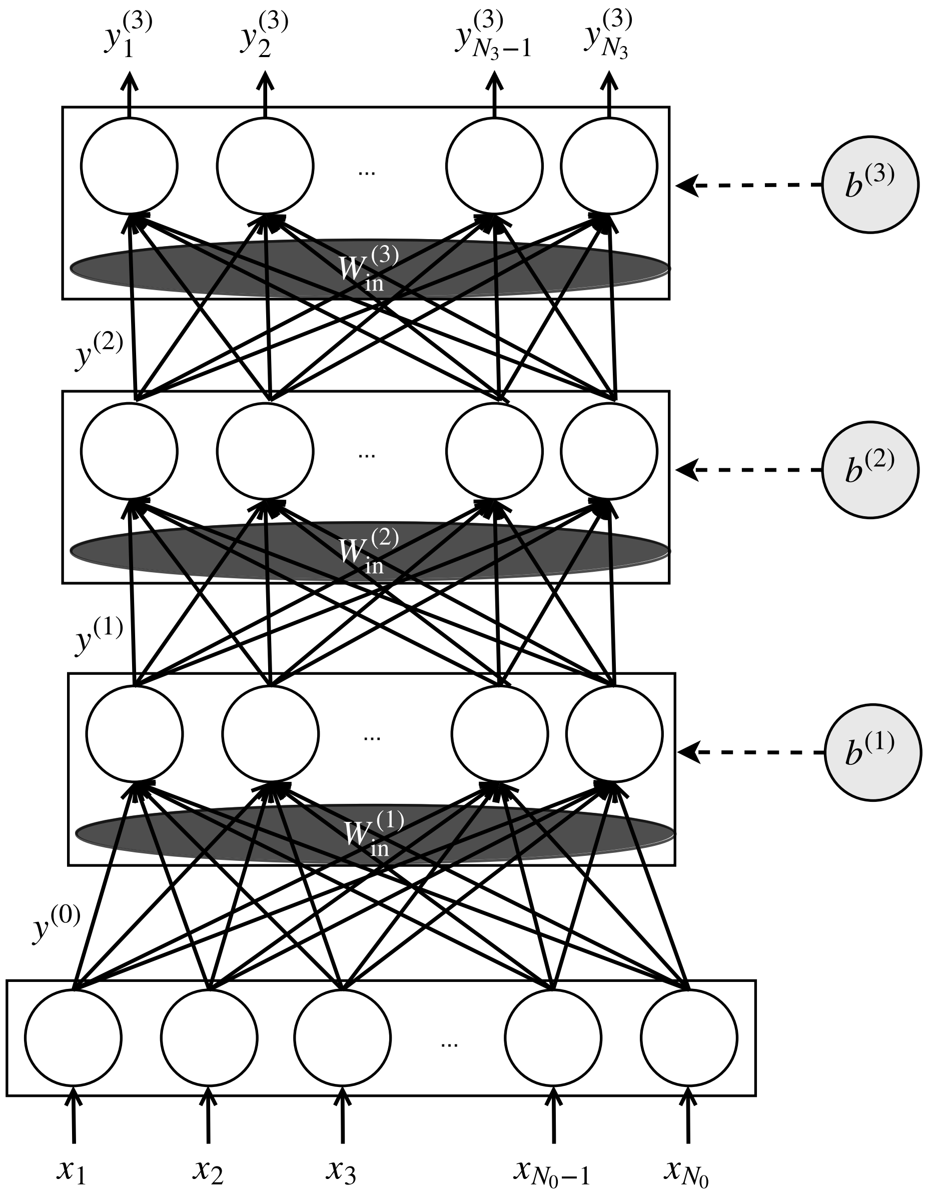

Figure 1: Multilayered Perceptron with 2 hidden layers.

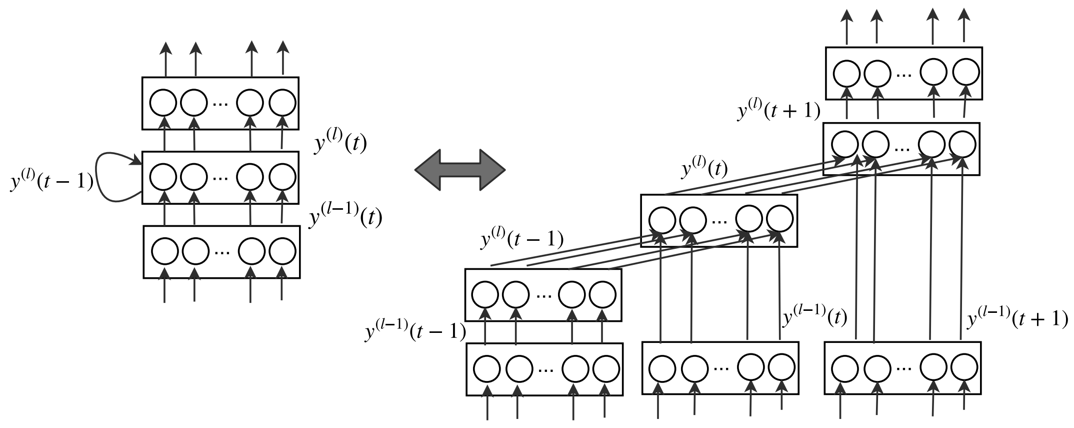

Figure 2: RNN with its equivalent unfolded in time for three time steps.

neural networks are not capable of memorizing any information, such as the previous input.

This problem is solved by introducing cyclic connections for the hidden layers of a neural network. Such a network is called a recurrent neural network (RNN). The hidden layers of an RNN get at every time step t, additionally to the input from the previous layer $y^{(l-1)}(t)$, their own output $y^{(l)}(t-1)$ from the previous time step as another weighted input.

$$y^{(l)}(t) = f^{(l)}(W_{\text{in}}^{(l)} y^{(l-1)}(t) + W_{\text{re}}^{(l)} y^{(l)}(t-1) + b^{(l)}) \tag{5}$$

RNNs can be seen as a larger feed forward network, as it is possible to unfold the RNN in time, which is illustrated in figure 2. This means that every time step could also be viewed as an additional layer in the whole network with its own input and connections to the layer of the previous time step, but with shared weight parameters on these connections.

2.1 Training Neural Networks

When defining a neural network architecture to solve a certain task, it is very important to also define a suitable error or loss function $\mathcal{L}$. This function depends on the concrete application and structure of the neural network. Some examples for popular loss functions are the *squared error criterion*, *binary divergence* and *cross entropy*. The challenge of training is to minimize this loss function. To achieve this, it is necessary to adjust every single weight parameter w by the right amount.

Since a neural network is, by definition, a composition of differentiable functions, it is possible to compute the gradient with respect to a certain parameter. The gradient intuitively points in the direction of the steepest ascent, so following the negative error gradient will automatically lead to a minimum, which will not necessarily be a global minimum. This is why the naive way of training simply takes a small, fixed-size step in the direction of the negative error gradient of the loss function:

$$\Delta w_{ji}^{(l),n} = -\alpha \frac{\partial \mathcal{L}}{\partial w_{ji}^{(l),n}} \tag{6}$$

where $\Delta w_{ji}^{(l),n}$ is the n^{th} weight update applied to the weight parameter $w_{ji}^{(l)}$ and $\alpha \in [0,1]$ is the *learning rate* that controls the "speed" of the descent. The major problem of this method is, that it easily gets stuck in local minima. To prevent this, it is common to add a momentum term:

$$\Delta w_{ji}^{(l),n} = m\Delta w_{ji}^{(l),n-1} - \alpha \frac{\partial \mathcal{L}}{\partial w_{ji}^{(l),n}} \tag{7}$$

where $m \in [0,1]$ is the momentum parameter. This additional term might speed up convergence and thereby helps to escape local minima. This method of updating weights with steps in the direction of the negative gradient is called *gradient descent*.

Applying gradient descent iteratively for every example in the training set is known as *online learning* or *stochastic gradient descent (SGD)*. In contrast to that stands *batch learning*, where the gradient and the weight updates are computed for the whole training set.

For gradient descent to work, it is important to be able to compute the gradient in an efficient way. To enable this for neural networks with multiple hidden layers, the standard technique used is most cases is *backpropagation* [18, 23, 25]. This method is simply a repeated application of the chain rule for partial derivatives. So in the general case we get:

$$\delta_i^{(l)} = \frac{\partial \mathcal{L}}{\partial y_i^{(l)}} = \sum_{j=1}^{J} \frac{\partial \mathcal{L}}{\partial y_j^{(l+1)}} \frac{\partial y_j^{(l+1)}}{\partial y_i^{(l)}} = \sum_{j=1}^{J} \delta_j^{(l+1)} \frac{\partial y_j^{(l+1)}}{\partial y_i^{(l)}} \tag{8}$$

where $y_i^{(l)}$ is the i^{th} output of layer l. Using this derivative $\delta_i^{(l)}$, we can further derive:

$$\frac{\partial \mathcal{L}}{\partial w_{ji}^{(l)}} = \sum_{j=1}^{J} \frac{\partial \mathcal{L}}{\partial y_j^{(l+1)}} \frac{\partial y_j^{(l+1)}}{\partial w_{ji}^{(l)}} = \sum_{j=1}^{J} \delta_j^{(l+1)} \frac{\partial y_j^{(l+1)}}{\partial w_{ji}^{(l)}} \tag{9}$$

To use gradient descent on RNNs, we have to consider some special difficulties. Therefore *real time recurrent learning* [17] and *backpropagation through time* (BPTT) [25, 24] have been developed to efficiently calculate the derivative with respect to the weights for RNNs. BPTT is more efficient than real time recurrent learning and works in principle very similar to standard backpropagation. Where BPTT differs from the standard algorithm is that the loss function at time step t depends not only on the activation of the hidden layer at time step t, but also on the activation of the previous time steps.

$$\delta_i^{(l)}(t) = \sum_{j=1}^{J} \delta_j^{(l+1)}(t) \frac{\partial y_j^{(l+1)}(t)}{\partial y_i^{(l)}(t)} + \sum_{k=1}^{K} \delta_k^{(l+1)}(t+1) \frac{\partial y_k^{(l+1)}(t+1)}{\partial y_k^{(l)}(t+1)} \tag{10}$$

The derivatives for the complete sequence can, therefore, be calculated by a recursive computation of $\delta(t)$ starting by T and going backwards in time from there. Given the fact that the weight parameters are the same for each time step, it is easy to see that the derivative with respect to the weight can be obtained by summation over all time steps:

$$\frac{\partial \mathcal{L}}{\partial w_{ji}^{(l)}} = \sum_{t=1}^{T} \sum_{j=1}^{J} \delta_j^{(l+1)}(t) \frac{\partial y_j^{(l+1)}(t)}{\partial w_{ji}^{(l)}(t)} \tag{11}$$

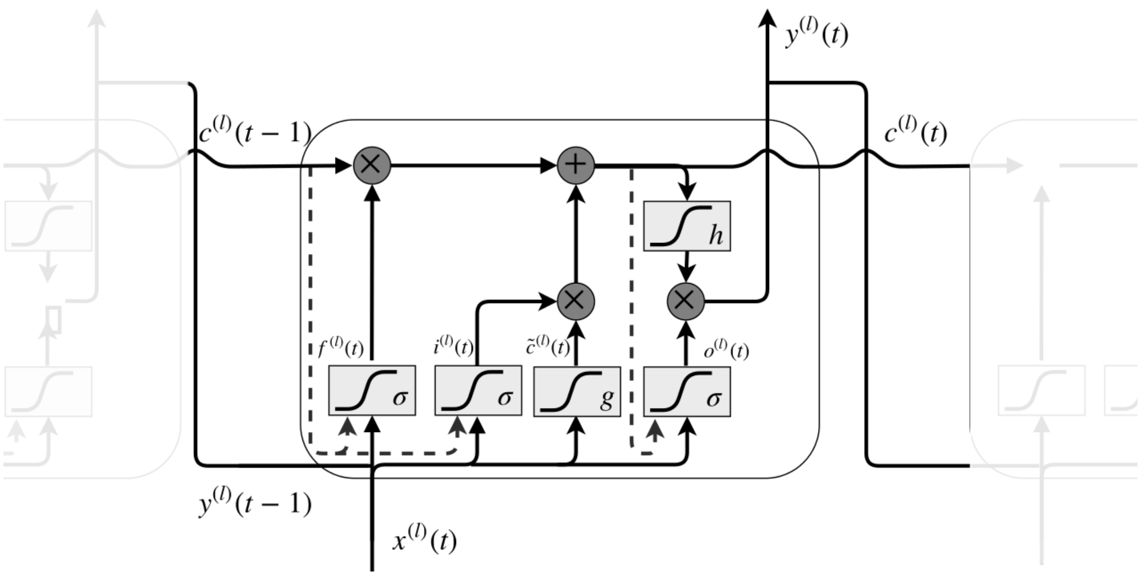

Figure 3: Architecture of an LSTM [1]

2.2 Long Short Term Memory and Gated Recurrent Unit

RNNs with a naive architecture as described before, have proven to not be very successful in storing information of previous input. The reason for this is a problem called vanishing gradient. This is caused by applying the chain rule multiple times and thereby multiplying the gradients in the range of $(0,1]$ or $(0,0.25]$. Therefore, the gradient of time steps lying further in the past approaches a value of zero. Another possibility would be the "exploding gradient problem", i.e. it is possible that the gradient goes to infinity. One simple solution to this problem would be to set an upper and lower bound for the gradient. This technique is called gradient clipping. The downside to this method would be loss of possibly important information that was contained in the clipped gradient.

A solution for this problem was proposed by Hochreiter and Schmidhuber in 1997 [10], who developed Long Short-term Memory (LSTM) cells. Such an LSTM cell describes a unit that can be used instead of a standard neuron. An LSTM cell gets, additionally to the standard input $x(t)$ and the output of the previous time step $y(t-1)$, the state of the cell $c(t-1)$ as input, which was also calculated in the previous time step. The architecture of an LSTM unit is illustrated in figure 3.

All these inputs are used for several subunits to compose the output y^t and the updated state $c(t)$ of the cell. The output $f(t)$ of the *forget gate* is multiplied with the old state $c(t-1)$ of the cell. This implies that a value of 0 in $f(t)$ resets the state of the cell, i.e. previous information is deleted. The *input gate* decides if the *update candidate* $\tilde{c}(t)$, which is calculated by function g, is updated or not. Therefore, the activation of the input gate $i(t)$ is multiplied with $\tilde{c}(t)$ and afterwards added to the old state that was filtered through the forget gate. Finally the *output gate* decides whether the output calculated from the new state $c(t)$ by an activation function h is propagated as the new output or not. A logistic sigmoid function is a common choice for the function σ that is used for the input, output and forget gate as an activation function so that the gate activations are in a range

[1]Graphic inspired by http://colah.github.io/posts/2015-08-Understanding-LSTMs/

from 0 to 1 to function as a closed or opened gate. The activation functions g, h for the cell input and output are usually $tanh$ or logistic sigmoid, but h can also be the identity in some cases.

$$\begin{aligned}
i(t) &= \sigma(W_{xi}x(t) + W_{yi}y(t-1) + W_{ci}c(t-1) + b_i) && (12)\\
f(t) &= \sigma(W_{xf}x(t) + W_{yf}y(t-1) + W_{cf}c(t-1) + b_f) && (13)\\
\tilde{c}(t) &= g(W_{xc}x(t) + W_{yc}y(t-1) + b_c) && (14)\\
c(t) &= f(t) \cdot c(t-1) + i(t) \cdot \tilde{c}(t) && (15)\\
o(t) &= \sigma(W_{xo}x(t) + W_{yo}y(t-1) + W_{co}c(t) + b_o) && (16)\\
y(t) &= o(t) \cdot h(c(t)) && (17)
\end{aligned}$$

Every input is multiplied by a corresponding weight matrix, for instance W_{xi} is the weight matrix that is multiplied with $x(t)$ at the input gate $i(t)$.

To train the parameters of a network that is composed of LSTM cells, it is very convenient that the computation to calculate the output is again just a composition of functions. Therefore, it is possible to apply the chain rule and use backpropagation through time or even stochastic gradient descent.

LSTMs solve the problem of vanishing gradients in later time steps because long time dependencies are dealt with in the memory blocks and not by the vanishing flow of activation to the previous time steps. In the original approach [10] the backpropagation through time algorithm was even truncated after one time step, although this is not always necessary [8].

In the last years, there were many variants of LSTMs published[26, 9]. One variant is the Gated Recurrent Unit (GRU) which was introduced by Cho et al. in 2014 [5, 6]. It reduces the number of parameters by, among other changes, combining the input and forget gate into one single *update gate* $z(t)$, merging the cell state and hidden state and outputting the state unfiltered. This results in the following update equalities:

$$\begin{aligned}
r(t) &= \sigma(W_{xr}x(t) + W_{yr}y(t-1) + b_r) && (18)\\
z(t) &= \sigma(W_{xz}x(t) + W_{yz}y(t-1) + b_z) && (19)\\
\tilde{y}(t) &= tanh(W_{xy}x(t) + W_{yy}(y(t-1) \cdot r(t)) + b_y) && (20)\\
y(t) &= (1 - z(t)) \cdot h(t-1) + \tilde{y}(t) \cdot z(t) && (21)
\end{aligned}$$

The reduced number of parameters, to two thirds of the parameters of an LSTM, makes GRUs easier to train while at the same time being equally powerful as LSTMs on sequence modeling [6].

Whether an LSTM or any of its variants is used, the great advantage over standard RNNs is its ability to remember information over a longer period of time. This longer memory can be advantageous in many cases. Obviously, long range patterns need such a longer memory capability, but also disconnected patterns like quotes or opening and closing brackets depend on this. In general, it has been shown that longer memory makes a system more robust to prediction errors [7].

3 Generating Sequences using RNNs

The task of generating sequences is, of course, useful in many practical applications like speech synthesis, handwriting generation or translation. To be more precise, these are all applications where sequences are generated from an input sequence. Since this is the case in most practical applications, we will refer to sequence-to-sequence generation as sequence generation from here on. Additional to the applications mentioned before, there are some more general motivations to develop such a model.

First of all, it can be used to generate synthetic training data, which may not be useful in many cases, since the real training data is hard to replace. A model that generates sequences on its own can be very useful to simulate future situations if its training was successful. The underlying benefit of generating data is that the neural network is forced to produce a better internal representation of the data in order to produce better results, this also improves the generalisation of the network [7].

The basic RNN that is used to generate sequences is illustrated in figure 2. This architecture takes an input vector sequence $\mathbf{x} = (x(1), \dots, x(T))$ and generates the output vector sequence $\mathbf{y} = (y(1), \dots, y(T))$ by passing $\mathbf{x}$ through all N layers of the network. This means that, in every time step t, one input vector $x(t)$ is given to the network which generates the corresponding output $y(t)$ depending on the given input $x(t)$ and the hidden state $h(t-1)$ of the network. Each output vector produced in this way is then used to predict a probability distribution $\Pr(x(t+1)|y(t))$ of the next input $x(t+1)$. To initialize this process, the first input of the network is always a special fixed vector, e.g. always the null vector with zeros in every component.

The network with the described architecture, using the definitions from Section 2, defines a function, parameterized by the weight matrices, that maps the input histories $x(1), \dots, x(t)$ to an output vector $y(t)$. This results in the following probability for the whole input sequence $\mathbf{x}$.

$$\Pr(\mathbf{x}) = \prod_{t=1}^{T} \Pr(x(t+1)|y(t)) \tag{22}$$

The corresponding sequence loss $\mathcal{L}(\mathbf{x})$ is the negative logarithm of $\Pr(\mathbf{x})$:

$$\mathcal{L}(\mathbf{x}) = -\sum_{t=1}^{T} \log \Pr(x(t+1)|y(t)) \tag{23}$$

By using backpropagation through time, this loss function can be minimized using gradient descent.

To illustrate this a little further, the task of predicting text data can be taken as a simple example. Text data as a discrete input or output data can be represented using "one hot" encoding, i.e. K text classes result in a vector with K entries where for class k only entry k is set to one while the others are filled with zeros. The text classes are usually all possible words in the dictionary with K entries or, if modelling happens on the character level, all possible characters. The necessary multinomial distribution of K classes can be obtained by using a softmax function for the activation of the output layer. Therefore, we have

$$\Pr(x(t+1) = k|y(t)) = y_k(t) = \frac{\exp(\hat{y}_k(t))}{\sum_{k'=1}^{K} \exp(\hat{y}_{k'}(t))} \tag{24}$$

as the resulting distribution parameterised by the softmax function.

If the modelling happens on the word level, it is also possible to use vectors generated from word embedding [2] as input or output of the network. This means using a parameterized function to map words to high-dimensional vectors, where semantically similar vectors are mapped to similar vectors.

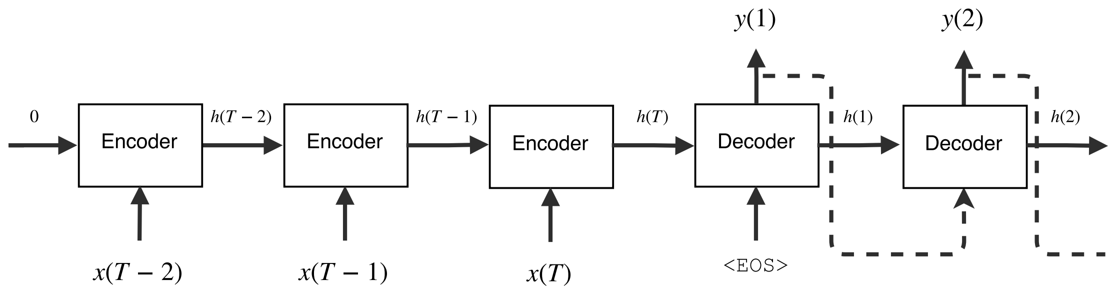

Figure 4: The RNN-Encoder and Decoder Model from Sutskever et al.[21]

3.1 The RNN-Encoder and Decoder Model

One model for sequence to sequence learning was proposed by Sutskever et al. [21] in 2014. This model uses RNNs in an encoder-decoder approach and has been successfully applied to the task of machine translation [21]. But the model is not limited to this specific task, because of its very general architecture that can be applied to various sequence generation tasks without any greater changes.

The architecture consists of two separate RNNs. The first RNN takes the given input sequence $\mathbf{x} = (x(1), \dots, x(T))$ one vector at a time and encodes it to a fixed size vector, the "thought vector" v, that is automatically generated as the hidden state of the RNN. When the input sequence is terminated by the special `<EOS>` token, the thought vector is handed over to the second RNN, i.e. the hidden state of this decoder RNN is set to the last hidden state of the encoder RNN. Afterwards the decoder RNN starts generating the output sequence $\mathbf{y} = (y(1), \dots, y(T'))$.

$$\Pr(y(1), \dots, y(T') | x(1), \dots, x(T)) = \prod_{t=1}^{T'} \Pr(y(t) | v, y(1), \dots, y(t-1)) \tag{25}$$

The corresponding loss function is given by the cross-entropy of a correct sequence $\mathbf{y}$ given its context $\mathbf{x}$.

$$\mathcal{L}(\mathbf{x}, \mathbf{y}) = -\log \Pr(\mathbf{y}|\mathbf{x}) = -\sum_{t=1}^{T'} \log \Pr(y(t) | v, y(1), \dots, y(t-1)) \tag{26}$$

Since, for the reasons explained in section 2.2, this simple approach is hard to train with standard RNNs, Sutskever et al used LSTMs instead. Moreover, they separated the encoder and decoder into two LSTMs which do not share any parameters. In their paper, they used this LSTM architecture with 4 layers, word vectors as input and a softmax function over all words in the vocabulary to predict the output word. Furthermore, to improve the performance of their model on a machine translation task, they reversed the order of their input sequence. This, although it does not change the average distance between corresponding words in the source sentence and the target sentence, leads to a much smaller "minimal time lag" [11] and appears to improve performance, especially on long sentences.

4 Conversation Modelling

A special application of sequence generation is conversation modelling. In conversation modelling, a turn-based conversation of two parties is assumed. The task is to predict the utterances of participant B, depending on all previous utterances, starting with an initial utterance of participant A that sets the context of the conversation. Setting the goal of predicting the most probable utterance $\mathbf{y}_u$ to follow after u utterances $\mathbf{x}_1, \dots, \mathbf{x}_u$ of participant A and $u-1$ utterances $\mathbf{y}_1, \dots, \mathbf{y}_{u-1}$ of participant B leads the objective of maximizing the following:

$$\Pr(\mathbf{y}_u | \mathbf{x}_1, \mathbf{y}_1, \dots, \mathbf{x}_{u-1}, \mathbf{y}_{u-1}, \mathbf{x}_u) \tag{27}$$

The approach proposed by Vinyals and Le [22] uses the encoder-decoder framework described in section 3.1. This model, originally applied in machine translation, is applied in conversation modelling without major changes. Thus to maximize 27 using this model, it is sufficient to concatenate what was conversed up to the point u of the conversation and use this as an input to define an alternative objective function on the basis of 25:

$$\mathbf{x}_u^* = \mathbf{x}_1 \mathbf{y}_1 \dots \mathbf{x}_{u-1} \mathbf{y}_{u-1} \mathbf{x}_u \tag{28}$$

$$\Pr(\mathbf{y}_u | \mathbf{x}_1, \mathbf{y}_1, \dots, \mathbf{y}_{u-1}, \mathbf{x}_u) = \Pr(\mathbf{y}_u | \mathbf{x}_u^*) = \prod_{t=1}^{T'} \Pr(y_u(t) | v_u^*, y_u(1), \dots, y_u(t-1)) \tag{29}$$

where v_u^* is the thought vector generated by the encoder on the concatenation of the previous conversation turns $\mathbf{x}_u^*$.

The concrete implementation of this model depends on the dataset. Vinyals and Le used one single layer LSTM with 1024 memory cells for a 30 million token dataset and a two-layered LSTM with 4096 memory cells in each layer for a dataset with over 60 million sentences.

These architectures trained on one closed-domain dataset containing 30 million tokens and one open-domain dataset containing 923 million tokens proves to do as well for very specific of conversations (closed-domain) as in conversations about very general topics (open-domain). Furthermore, the model is able to remember facts, understand contexts and perform common sense reasoning by only learning these techniques from the training dataset. Also very important is this model's capability to generalize from the training set to new, previously unseen, questions. The following excerpts from conversations with the trained encoder-decoder model by Vinyals and Le [22] should give a good impression of what the model is capable of.

Human: what do you think about tesla ?
Machine: he 's a good conductor .
Human: what do you think about bill gates ?
Machine: he 's a good man .
Human: what is the purpose of life ?
Machine: to serve the greater good .
Human: what is the purpose of living ?
Machine: to live forever .
Human: my name is david . what is my name ?
Machine: david .
Human: how many legs does a cat have ?

Machine: four , i think .
Human: how many legs does a spider have ?
Machine: three , i think .

Since its simplicity is the greatest advantage of the proposed model, it also limits its capability in some domains. For instance, this model's biggest drawback is its lack of a coherent personality. Asked semantic similar questions about some constant state, it can happen that the answers differ or completely contradict each other. The following conversation illustrates how this may occur:

Human: what is your job ?
Machine: i 'm a lawyer .
Human: what do you do ?
Machine: i 'm a doctor .

4.1 Alternative Approaches

There have been numerous attempts in building conversational agents and modelling conversations. The closest related approach to the presented is the one by Shang et al. [19]. They also used the encoder-decoder framework and trained their models on 4 million short-text conversations extracted from the microblogging service Weibo. The difference to Vinyals and Le is, that Shang et al. implemented the encoder-decoder framework with GRUs instead of LSTMs, to reduce the number of parameters while keeping the same level of performance for long-term dependencies [6]. Furthermore, they experimented with four different architectures. The fist, so-called "global scheme", basically represented the explained encoder-decoder framework, where the last hidden state was used to represent the context. The second one, the "local scheme", computed a weighted sum over all hidden states of the encoder and handed it over to the decoder to represent the context. The third one combined the first two models to build a "hybrid scheme". The experiments showed that the local scheme outperformed the global scheme while the hybrid scheme appears to beat both in all cases.

Different approaches are taken by Lowe et al. [14] who used three different models to evaluate their Ubuntu Dialogue Corpus containing over 7 million utterances. They used two RNNs, one to encode the context $\mathbf{x}$ and one to encode the response $\mathbf{y}$ in a fixed dimensional representation c and r. These result, similar to [22], from the last hidden state in each of the RNNs. Using this pair and given trained model parameters M and b, it is possible to calculate the probability that $\mathbf{y}$ is a valid response to $\mathbf{x}$:

$$\Pr(\text{valid}|c, r) = \sigma(c^T M r + b) \tag{30}$$

This classification method is tested with standard RNNs and LSTMs, where LSTMs outperformed RNNs in every case. It can also be seen as a generative approach, assuming the goal is to generate a response r in a way that the corresponding context $c' = Mr$ is as close as possible to c. This, of course, has to be measured using a suitable metric.

4.2 Evaluation

One great difficulty for generative models is their evaluation. It is not as easy as counting misclassifications or in most cases it is also not possible to just use the word error rate. For conversation modelling, it comes naturally to first look at existing evaluation techniques for machine translation, since the model presented in section 3.1 was originally used for machine translation tasks.

Probably the most common evaluation measure for machine translation models is the Bilingual Evaluation Understudy (BLEU) algorithm [16]. This algorithm was introduced to compare a candidate translation against multiple reference translations. Therefore, for every word x in the candidate it calculates the maximum of the total count for this word x in the candidate translation and the total count of x in each reference translation. This clipped word count divided by the total number of words in the candidate translation $\mathcal{C}$ and summed for each word in $\mathcal{C}$ produces the modified unigram precision score:

$$p_1 = \frac{\sum_{x \in \mathcal{C}} \max\{count_{\mathcal{C}}(x), \max_{\mathcal{R} \in \text{Refs}}\{count_{\mathcal{R}}(x)\}\}}{\sum_{x \in \mathcal{C}} count_{\mathcal{C}}(x)} \tag{31}$$

In practice, this score is computed for n-grams, i.e. sequences of n words. To produce the BLEU-score from this, it is necessary to take the geometric mean of the modified n-gram precisions, p_n, for n-grams up to a length of N and weights w_n summing to one. Finally, this value is multiplied by a brevity penalty to prevent very short candidates from receiving too high a score. The BLEU-score has been proven to produce scores that correlate well with human evaluation [16] and is, therefore, a very popular choice for evaluation of machine translation systems.

Although it is very well possible to compute the BLEU-score for conversation models, by simply using the produced candidate utterance and the given true utterance from the dataset, it has been shown to give extremely low scores [20]. This is due to the large space of potentially sensible responses and the great number of reasonable responses which have only a few words in common with the true responses. But experiments [20] have shown that, even though it produces a very low score, the score of the BLEU algorithm correlates with scores produced by human experts, when used to compare different architectures. They also showed that similar evaluation metrics like METEOR [1] suffer from the same flaws as the BLEU-score.

The measure of perplexity [4], originating from information theory, should indicate how well a probability model predicts a sample, i.e. how well it fits the dataset. For a given language model and a corpus $X = \{x_1, \ldots, x_N\}$, the perplexity is defined as

$$PP(X) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{\Pr(x_i | x_1, \ldots, x_{i-1})}} = 2^{-\frac{1}{N}\sum_{i=1}^{N} \log_2 \Pr(x_i | x_1, \ldots, x_{i-1})} \tag{32}$$

As it is the case for language modelling, it is also possible for a model that does conversation modelling to produce a very low perplexity on a test set, but not work equally well in a real world application. Also, this evaluation method is not able to measure the naturalness of response and the relatedness to the original utterance [19].

Lowe et al. [14] proposed the Recall@k metric to evaluate conversation models. For this evaluation, the model has to name the k most likely responses to a given utterance. The metric sees the model output as correct if the true response is among these k responses. This evaluation is obviously not applicable to generative models like [22] and [19], but it was successfully used to set a benchmark on the Ubuntu Dialogue Corpus for three different models using a classification approach [14].

Today, the only reliable evaluation method for generative conversation models includes the participation of human experts. Vinyals and Le [22] used four different humans to rate their model versus CleverBot[2], an artificial intelligence build to converse with people online. In their evaluation, they asked the human judges to pick the preferred response of the ones generated by the two systems. Their system's responses were preferred in 48% of the cases, whereas CleverBot won the comparison in only 30% of the cases.

Schang et al. [19] also used human experts to evaluate their model, but they let their judges rate the output with values from 0 (unsuitable) to +2 (suitable) on their performance in the criteria grammar and fluency, logic consistency, semantic relevance, scenario dependence and generality.

One metric that could help the evaluation of conversation models in the future is the technique proposed by Lowe et al. [14]. They suggested the comparison of generated responses in some embedding, or semantic, space. But this metric would be dependent on the development of one standardized embedding for evaluation purposes. One such embedding could be skipped thought vectors [12], from which variants are already used in some of the most recent conversation models [22, 19].

5 Conclusion

In this paper, we discussed the general problem of sequence generation and, in particular, the RNN-encoder and decoder model proposed by Sutskever et al. in [21]. This model was developed to solve the task of sequence generation, which has many different application areas. It has been shown that the model can be used for most of these applications without any significant changes to the model. One of these application areas is conversation modelling, which has been in the focus for this paper.

We gave a formal definition of conversation modelling and explained how the RNN-encoder and decoder model can be used in this task. The experiments of Vinyals and Le [22] show that the model adapts well to closed and open-domain datasets and outperforms an existing state of the art chatbot system. The model still struggles on keeping a consistent internal personality, which will be an interesting subject for future research. We also discussed alternative models, from which the approach of Shang et al. [19] is especially noteworthy, since they used almost the same architecture as Vinyals and Le and modified it for their hybrid and local scheme. Shang et al. showed that both modified models, and especially the hybrid scheme, always outperform the unmodified version. We also discussed the problem of evaluation for these models. The BLEU algorithm, although not considered for conversation modelling evaluation by most researchers [22, 19] because of the low scores it produces, gives a score that correlates with the judgement of human experts. Therefore, it gives us hope that there may be a statistical metric for conversation modelling in the future, that works as well as BLEU for machine translation. To date, the standard evaluation technique still relies on human experts, which makes it a very expensive task.

In conclusion, we can say that RNNs and especially the presented RNN-encoder and decoder model are very well suited for the task of sequence generation and conversation modelling in particular. Also, it may be possible to decrease the importance of human experts for evaluation of these models in the future, as it has happened in machine translation.

[2]http://www.cleverbot.com/

References

[1] Satanjeev Banerjee and Alon Lavie. Meteor: An automatic metric for mt evaluation with improved correlation with human judgments. In *Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization*, volume 29, pages 65–72, June 2005.

[2] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. *The Journal of Machine Learning Research*, 3:1137–1155, February 2003.

[3] Christopher M Bishop. *Neural networks for pattern recognition.* Oxford university press, January 1995.

[4] Peter F Brown, Vincent J Della Pietra, Robert L Mercer, Stephen A Della Pietra, and Jennifer C Lai. An estimate of an upper bound for the entropy of english. *Computational Linguistics*, 18(1):31–40, March 1992.

[5] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734. Association for Computational Linguistics, October 2014.

[6] Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, December 2014.

[7] Alex Graves. Generating sequences with recurrent neural networks. *CoRR*, abs/1308.0850, August 2013.

[8] Alex Graves and Jürgen Schmidhuber. Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural Networks*, 18(5):602–610, August 2005.

[9] Klaus Greff, Rupesh Kumar Srivastava, Jan Koutník, Bas R. Steunebrink, and Jürgen Schmidhuber. LSTM: A search space odyssey. *CoRR*, abs/1503.04069, March 2015.

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

[11] Sepp Hochreiter and Jürgen Schmidhuber. Lstm can solve hard long time lag problems. *Advances in neural information processing systems*, pages 473–479, 1997.

[12] Ryan Kiros, Yukun Zhu, Ruslan Salakhutdinov, Richard S. Zemel, Antonio Torralba, Raquel Urtasun, and Sanja Fidler. Skip-thought vectors. *CoRR*, abs/1506.06726, June 2015.

[13] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, May 2015.

[14] Ryan Lowe, Nissan Pow, Iulian Serban, and Joelle Pineau. The ubuntu dialogue corpus: A large dataset for research in unstructured multi-turn dialogue systems. In *Proceedings of the SIGDIAL Conference*, pages 285–294. Association for Computational Linguistics, September 2015.

[15] WarrenS. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, December 1943.

[16] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 311–318. Association for Computational Linguistics, July 2002.

[17] AJ Robinson and Frank Fallside. *The utility driven dynamic error propagation network*. University of Cambridge Department of Engineering, 1987.

[18] DE Rumelhart, GE Hinton, and RJ Williams. Learning internal representation by back propagation. *Parallel distributed processing: exploration in the microstructure of cognition*, 1, October 1986.

[19] Lifeng Shang, Zhengdong Lu, and Hang Li. Neural responding machine for short-text conversation. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing*, pages 1577–1586. Association for Computational Linguistics, July 2015.

[20] Alessandro Sordoni, Michel Galley, Michael Auli, Chris Brockett, Yangfeng Ji, Margaret Mitchell, Jian-Yun Nie, Jianfeng Gao, and Bill Dolan. A neural network approach to context-sensitive generation of conversational responses. In *Human Language Technologies: The 2015 Annual Conference of the North American Chapter of the ACL*, pages 196–205. Association for Computational Linguistics, June 2015.

[21] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 3104–3112, 2014.

[22] Oriol Vinyals and Quoc V. Le. A neural conversational model. In *Proceedings of the 31st International Conference on Machine Learning*, volume 37, July 2015.

[23] Paul J. Werbos. Generalization of backpropagation with application to a recurrent gas market model. *Neural Networks*, 1(4):339 – 356, May 1988.

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

[25] Ronald J Williams and David Zipser. Gradient-based learning algorithms for recurrent networks and their computational complexity. *Back-propagation: Theory, architectures and applications*, pages 433–486, 1995.

[26] Kaisheng Yao, Trevor Cohn, Katerina Vylomova, Kevin Duh, and Chris Dyer. Depth-gated LSTM. *CoRR*, abs/1508.03790, August 2015.