CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Decoding neural machine translation using gradient descent

Master's thesis in Computer Science, Algorithms, Language and Logic

Emanuel Snelleman

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2016

Master's thesis 2016:NN

Decoding neural machine translation using gradient descent

Emanuel Snelleman

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
Gothenburg, Sweden 2016

Decoding neural machine translation using gradient descent
Emanuel Snelleman

Supervisor: Mikael Kågebäck, Department of Computer Science and Engineering
Examiner: Krasimir Angelov, Department of Computer Science and Engineering

Master's Thesis 2016:NN
Department of Computer Science and Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2016

Decoding neural machine translation using gradient descent
Emanuel Snelleman
Department of Computer Science and Engineering
Chalmers University of Technology

Abstract

Neural machine translation is a resent approach to machine translation. Neural machine translation uses an artificial neural network to learn and perform translations. Usually an encoder-decoder architecture is used for the neural network. The encoder encodes the input sentence into a fix sized vector meant to represent the meaning of the sentence, the decoder then decodes that vector into a sentence of the same meaning in the other language. The decoder will generate the translation sequentially, and will output a probability distribution over the known words at each step of the sequence. From these probability distributions words needs to be chosen to form a sentence. The word chosen given any specific probability distribution will affect how the next probability distribution is generated. It is therefore important to chose the words well in order to get a translation as good as possible. In this thesis a way of choosing the words using gradient descent is tested. Decoding using gradient descent is compared to some other methods of decoding but gives no clear indication of working better than the other decoding methods. However, some results indicate gradient descent decoding might have some potential if further developed.

Keywords: machine translation, artificial neural network, gradient descent, beam search.

Acknowledgements

I would like to give a big thanks to my supervisor, Mikael Kågebäck. He was the one who came up with the idea for the thesis and he helped out a lot throughout the work on the thesis. I would also like to thank my examiner, Krasimir Angelov, for his feed back on the thesis.

Emanuel Snelleman, Gothenburg, August 2016

Contents

List of Figures

List of Tables

1
Introduction

Machine translation is translation, from one language to another, carried out by a computer. Neural machine translation which is machine translation based on artificial neural networks is a newly emerging approach to machine translation [1, 13]. In contrast to traditional phrase based systems [6], which consists of many small sub-components, each individually tuned, neural machine translation works by having a single neural network trained to read a sentence and output a translated sentence.
When the network is trained, it is trained to generate a probability distribution over the the words the network knows, for each word in the output sentence. When the network is generating (decoding) the translation the probability distribution for the words in the sentence are dependent on both the input sentence and also the previously generated words in the output sentence. This means a word needs to be chosen from the probability distribution and feed back to the network to generate the next word in the output sentence. One way to do this can be to simply choose the most probable word from the distribution at each step. This can give good results but choosing a less probable word in the beginning might result in a better translation as it will affect the probability distribution of the subsequent words. To get as good of a translation as possible it is desired to maximize the probability of the sentence according to the probability distributions. To maximize the probability a search can be performed to find the most probable sequence of words. Since the network can be taught a lot of words a full search might not be feasible. To do an approximate search, beam search is commonly used [1, 13].
For the network to process different words a special vector representation is used for the words. In this thesis, we will try to utilize the fact that the gradient of the probability of the output sentence can be calculated. Using this gradient, the direction that the vector representation, for the output words, should be changed in order to maximize the probability is given. Following the gradient seems likely to give highly probable translations faster than if beam search were to be used.

1.1 Goals

The aim of this thesis is to utilize gradient descent to optimize the probability of the output sentence according to the probability distributions that an artificial neural network has learned. Artificial neural networks can only process fixed size vectors so words have a certain vector representation when processed by the network. A problem may arise when using gradient descent to adjust the word vectors, since the resulting vectors may not match any vector representing a word. The gradient will

be calculated based on the joint probability of the words in the generated sentence. In order to solve the problem of vectors not matching any word representations terms will be added to to the function from which the gradient is calculated. These terms are then supposed to give an error when the vectors are far from the word representations. The idea is that this addition will regularize the vectors towards the word representations. In short the goals for this thesis are:

- Develop an error function for gradient descent decoding.
- Evaluate gradient descent decoding.

1.2 Related works

There are a lot of work that has been done on neural machine translation lately. One simple, yet effective way to apply neural networks to machine translation is to rescore the best results from some other good translation model [8].
There has also been a lot of work done on using a neural network for the complete translation task. Often in these cases an encoder-decoder architecture has been used, as in this thesis. A problem with that model is that it can perform badly on longer sentences. Bahdanau et al. [1] tries to improve neural machine translation for longer sentences. In their paper they conjecture that encoding the source sentence to a fixed length vector is a bottleneck. They instead encode the source sentence with a vector for every word in the source sentence. This is done by for every word producing a vector by encoding up to that word and encoding the sentence backwards to that word. Every such vector will then represent the whole sentence but with focus on the words close to the word the vector stems from as they will have contributed to the encoding more recently. The decoder can then learn to use these vectors when decoding and is not constrained to using a fixed length vector to represent the meaning of the whole source sentence.
Another problem with translation models based on neural networks, is that the models can be quite limited in the number of words they know. Luong et al.[7] tries to address this problem by first applying a word alignment algorithm to the training data. The network can then learn to, for every unknown word by the network, give the position of the corresponding word in the source sentence. Then in a post processing step every unknown word can be directly translated from the corresponding word in the source sentence, using a dictionary.
Another work related to this thesis is Leon A. Gatys et al. [3]. In their article they describe a way to combine the style of one image with the content of another image. This is done by using a convolutional network for object recognition. A convolutional network splits its calculations in to smaller units in several layers. Each such unit can be seen as an image filter that extracts some feature of the input image. These units, also called feature-maps will bring fourth important information about the content of an image while losing information about specific pixel values. Information about the style of an image is given by correlations in the feature-maps. To generate an image with the style of a certain image and the content of another image, a white noise image is feed to the the network. Gradient descent is then used to find an image that gives a similar output to the two original images in some of the

feature-maps. To match the content of the content image, gradient descent is used to alter the white noise image so that it gives a similar output to the content image, in some feature-maps. To match the style of the style image, gradient descent is used to alter the image so that it gives similar correlations as the style image in the feature-maps.

2
Background

In this section some significant background theory will be explained. For the most part theory behind artificial neural networks, that will be used for the translation model, will be explained. There are also two parts regarding the evaluation of the gradient descent decoding, BLEU and beam search.

2.1 Artificial neural networks

An artificial neural network is a model in machine learning inspired by biological neural networks. They are generally presented as interconnected neurons exchanging messages between each other.

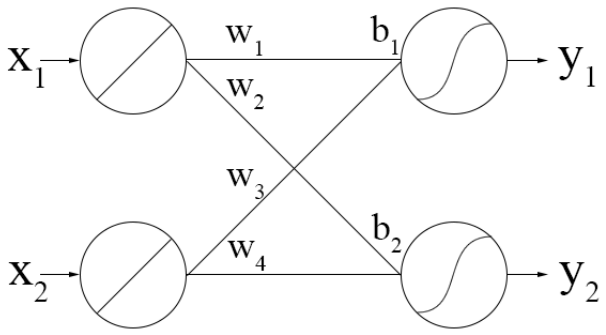

Figure 2.1: Illustration of a one layer network

Figure 2.1 shows a simple one layer network, with two input neurons and two output neurons.

$$y_1 = g(w_1 x_1 + w_3 x_2 + b_1) \tag{2.1}$$

$$y_2 = g(w_2 x_1 + w_4 x_2 + b_2) \tag{2.2}$$

Equation 2.1 and 2.2 shows how the output y_1 and y_2 are calculated. Usually the input and output are seen as vectors, as $\{y_1, y_2\}$. In the equations g is whats called an activation function. Usually more layers are added to the network, in that case the activation function should to be a nonlinear function.

When adding more layers and neurons, neural networks can solve many tasks, such as face recognition [10], speech recognition [12], and translation [1, 13]. Generally neural networks are taught to solve such tasks by learning from examples of the task where the output is known. This means that the weights and biases of the network, w and b in Figure 2.1, are altered until the network solve the tasks accurately according to the known solutions.

2.1.1 Sigmoid function

The sigmoid function can be used as an activation function for neural networks. By applying the sigmoid function to the value of a neuron the output of that neuron will always be between zero and one. The sigmoid function is defined by the formula

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

2.1.2 Softmax

Softmax can also be used as an activation function. It is often used if the output of a layer is supposed to represent a probability distribution.

$$s(x)_i = \frac{\mathrm{e}^{x_i}}{\sum_j \mathrm{e}^{x_j}} \tag{2.4}$$

By applying the softmax function to a vector, all elements of the vector will have values between zero and one, and will sum to one.

2.1.3 Training

Neural networks can be trained using backpropagation of errors [11]. The errors are some measurement of how far the networks output is from the desired output (the function calculating the errors is called the error function). Gradient descent is used to update the weights and biases of the network, according to the gradient of the error function, so that the errors are minimized. For the network in Figure 2.1 the derivative used to update w_1, given an error function E and a desired output $\hat{y}_1$, would be the following

$$\frac{\partial E}{\partial w_1} = E'(\hat{y}_1, g(w_1 I_1 + w_3 I_2 + b_1)) * g'(w_1 x_1 + w_3 x_2 + b_1) * x_1 \tag{2.5}$$

In Equation 2.5 the chain rule has been applied twice, once for the error function E and once for the activation function g. When more layers are used in the network the chain rule will have to be applied on the activation function for every layer.
Usually a learning rate parameter is used, set to a value between zero and one. This parameter is multiplied with the gradient so that any specific error does not affect the network to much, instead several examples are used so that the network learns a little bit from every example. Often the learning rate is decreased over time while training so that the training can get gradually more fine tuned. Reducing the learning rate is called learning rate decay and can be done in different ways so that

the learning rate for example decays exponentially or linearly. A set of data for learning can be trained upon several times, a run through all training data is called an epoch.
Gradient descent can be used with momentum, momentum causes the gradient descent to increase progress in dimensions where the error function changes little, and decrease the amount of oscillations in dimensions where error function changes the sign of its derivative. This is done by remembering the previous update of a weight or bias and increasing the next update by a fraction of the last update. How much of the last update that is used is determined by a parameter.

2.1.3.1 Cross entropy

Cross entropy is a measure in information theory that is based on two different probability distributions over the same set.

$$H(p, q) = -\sum_{i}(p_i \log q_i) \tag{2.6}$$

p and q are different probability distribution.

When used with neural networks, cross entropy gives a good way to measure how closely the calculated probability distribution reflects the target probability distribution. It can be used as an error function together backproppagation when training networks that should learn to output probability distributions. I this case p would be the desired distribution and q the distribution predicted by the network.

2.1.4 Recurrent neural network

A recurrent neural network (RNN) is a network where neurons are connected in such a way that a directed cycle is formed. This creates an internal state in the network that can be seen as a memory. This memory allows an RNN to process sequences of inputs. This is beneficial in for example natural language processing as arbitrary length sentences can be processed word by word and the internal memory state allows the network to learn things about the whole sentence. In addition to translation, RNNs have been used for tasks such as handwriting recognition [4] and speech recognition [12].

2.1.4.1 GRU

A GRU (gated recurent unit) is a sort of RNN introduced in [2]. Compared to a vanilla RNN a GRU has a gated hidden state. This is to counteract the vanishing gradient problem that vanilla RNNs suffer from. The vanishing gradient problem is a problem that arises when training neural networks with backpropagation. The problem is that for every layer, the calculations for the gradients, decreases the gradients amplitude. This because of the way the chain rule gets applied to the activation functions several times for multi-layered networks. The gradient of commonly used activation functions are usually in the range of $(-1, 1)$ or $[0, 1)$. For a network with n layers or an RNN that learns an n long sequence, these small

numbers will be multiplied at least n times. This means that a network with many layers or an RNN that learns long sequences the vanishing gradient may lead to the foremost layers being trained very slowly. If activation functions that can have larger gradients are used the exploding gradient problem might be encountered instead. A GRU calculates its hidden state and output as follows

$$z = \sigma(x_t U^z + h_{t-1} W^z) \tag{2.7}$$

$$r = \sigma(x_t U^r + h_{t-1} W^r) \tag{2.8}$$

$$\tilde{h}_t = tanh(x_t U^h + (h_{t-1} \odot r) W^h) \tag{2.9}$$

$$h_t = (1 - z) \odot \tilde{h}_t + z \odot h_{t-1} \tag{2.10}$$

In the equations above $\odot$ indicates element wise multiplication and σ is the sigmoid function. x_t is the input and h_t is the output and hidden state of the GRU at time t. U and W are trainable weights for r, z, and h.

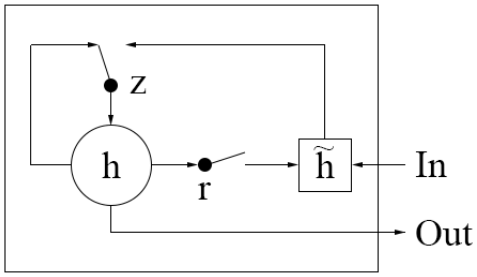

Figure 2.2: Illustaration of a GRU

Figure 2.2 illustrates a GRU unit. The reset gate r lets the GRU ignore the previous hidden state and forget irrelevant information. The update gate z lets the GRU select whether the hidden state will be updated with the new hidden state $\tilde{h}_t$.

2.1.5 RNN encoder-decoder

Many neural machine translation systems follow an encoder-decoder architecture [2, 1, 13]. An encoder-decoder network is based on two RNNs. The encoder RNN reads a sequence and encodes it into a fixed size vector. The decoder then takes this vector and decodes it in to a new sequence. The encoded fixed size vector is taken from the internal memory state of the encoder after it has processed the entire input sequence. The decoder uses that vector as its initial internal state when it starts generating the output sequence. The input to the decoder RNN is based on the previously generated output, for the first input a special start symbol can be used. When the encoder-decoder architecture is used for translation it can be seen as reading the input sentence and encode it into a vector representing the meaning of the sentence. The decoder then generates a sentence in the target language

with the same meaning. When the network is trained to translate, the decoder is trained to output a probability distribution over the words in the target language. This probability distribution will then be dependant on the memory state from the encoder and the earlier words fed as input to the decoder.

2.2 Machine translation

The problem of translating natural language can be seen as finding a target sentence T that maximizes the conditional probability of T given a source sentence S, Equation 2.11. In neural machine translation the probability distribution is modeled and learned by an artificial neural network. As the probability of every word in the target sentence $\{t_1, t_2, \cdots, t_n\}$ depends on the previously predicted words. The probability of the sentence is given by the joint probability of every word given the previous words and the source sentence, Equation 2.12.

$$\arg\max_{T} P(T|S) \tag{2.11}$$

$$P(T|S) = \prod_{n} P(t_n|t_1, \cdots, t_{n-1}, S) \tag{2.12}$$

2.3 Word representation

Artificial neural networks can only handle fixed size input and output vectors. This means that if the input and output of the network are supposed be words, those words needs to be represented as fix sized vectors. One way to represent words for artificial neural networks are as one-hot vectors. A one-hot vector is a vector where one element is one and the rest are zero. Representing words as one-hot vectors has the benefit of not assuming anything about the correlation between words, as all words are equally distanced from each other. A hidden layer in the network can be used so that the network can learn the correlation between the words by itself. Since the input and output vectors must be a fixed size, the one-hot representation limits the amount of words the network can understand. The words the network does know are collectively called a vocabulary, which often also include symbols such as dots and commas as well. The words and symbols are collectively called tokens. The vocabulary is usually build from the most common tokens from the training data. When training and using the network any token not in the vocabulary is usually represented with a certain one-hot vector representing an unknown token.

2.4 Beam search

Beam search is a search algorithm similar to breadth first search. At each level of the tree all succeeding nodes are generated but only a predetermined number of nodes are stored and expanded in the next iteration, based on some cost for the nodes. As some nodes are pruned that could potentially be part of the optimal solution,

beam search is not optimal. Beam search can be used to improve the performance of machine translation models by attempting to solve Equation 2.11. When the distribution of the first translated word is generated, the k most probable words can be used to generate the next distribution in the sequence. Then the k most probable translation so far according to the joint probability, as in Equation 2.12, is used to generate the next distribution and so on.

2.5 Europarl

Eurparl is a parallel corpus for statistical machine translation extracted from the proceedings of the European Parliament [5]. The corpus includes parallel text in 11 different languages. Parallel text means text paired with its translation into another language. The corpus is for exampled used in the workshop on statistical machine translation (WMT). In this thesis the French-English corpus will be used, which contains over 22 million English sentences paired with their French translation.

2.6 BLEU

BLEU (bilingual evaluation understudy) is a method for automatic evaluation of machine translation [9]. Human evaluation may be very extensive and accurate but can be costly and time consuming. Automatic evaluation can therefore be very useful. The idea behind the BLEU evaluation is that the closer a translation is to a human translation the better it is. BLEU gives a score depending on a measure of how close the translation is to one or several human reference translations.

3
Model

This section will describe the translation model used in this thesis. It will also explain gradient descent decoding and the error function used for the gradient descent decoding.

3.1 Translation model

The translation model used is based on the simple encoder-decoder design as introduced in [2]. As described in Section 2.1.5 the encoder reads the source sentence word by word and encodes it in to a fixed size vector, the decoder then generates a translated sentence based on the encoding.

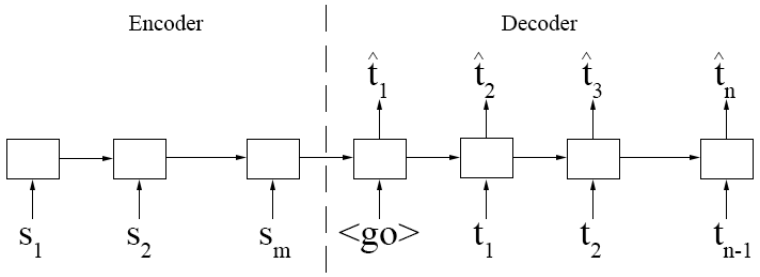

Figure 3.1: Visualisation of an encoder-decoder network

Each of the boxes in Figure 3.1 represents the RNN cells at the different time steps of processing the input and generating the output. For the encoding part of the network the only interesting part is the internal memory of the encoders RNN cell and the output of the RNN cell is ignored. After the last input has been processed by the encoder, the internal memory of the encoders RNN cell is extracted and used to initialize the internal memory of the decoders RNN cell. The first input to the decoder is a special go symbol, with that it generates a probability distribution over all known tokens, $\hat{t}_1$. From that distribution one token is chosen, t_1, and fed back to the decoder to generate the probability distribution for the next token. It is the chosen tokens, $t_1, t_2, \cdots t_n$, that form the translated sentence.
Before the word vector is processed by the RNN cell it is embedded into a continuous space vector representation. This continuous space representation allows for

the network to learn correlations between words that the one-hot representation intentionally does not include. The embedding is performed by a linear hidden layer between the the input and RNN cell and learns correlation between words when the network is trained. The encoder and decoder has separate embedding layers. The decoder also has a layer after the RNN cell, taking the output from the continuous space to the probability distribution over words in the target language. To make sure the output vector becomes a probability distribution softmax is applied to the output of the last layer.
The RNN cell used for both the encoder and decoder is a GRU, described in Section 2.1.4.1.
As Sutskever et al. does in [13] the input to the encoder will be reversed. In their paper they explain that feeding the input backwards reduces the number of long term dependencies and they show that it gives them better results.

3.2 Gradient descent decoding

The gradient descent decoding is meant to optimize Equation 2.12 similar to decoding with beam search. It will use gradient descent and backpropagation in order to find the input to the decoder that maximizes Equation 2.12. Similar to when a neural network is trained with gradient descent, the gradient descent might have to be applied several times.
Assuming $\{t_1, t_2, \cdots, t_n\}$ are one-hot vectors and $\{\hat{t}_1, \hat{t}_2, \cdots, \hat{t}_n\}$ are the probability distributions which the network generates, Equation 2.12 can be written as:

$$P(T, \hat{T}) = \prod_n (t_n \hat{t}_n) \tag{3.1}$$

Using log probability allows to rewrite the product as a sum and negating it allows for minimizing rather then maximizing.

$$H(T, \hat{T}) = -\log P(T, \hat{T}) = -\sum_n (t_n \log \hat{t}_n) \tag{3.2}$$

Equation 3.2 is the same as the cross entropy of T and $\hat{T}$ and is the primary part of the error function used together with the backpropagation. A potential problem is that when gradient descent alters the vectors $\{t_1, t_2, \cdots, t_n\}$ in order to minimize H, the vectors might not be one-hot vectors. This could cause a problem as the network will be trained on one-hot vectors. This could potentially be solved by adding additional terms to the regularize the target vectors towards one-hot vectors.

3.3 One-hot regularization

The gradient calculated for the gradient descent decoding will be a value in a continuous space and so gradient descent will yield values in a continuous space. Since the words are discrete and represented as one-hot vectors, the values given by gradient descent might not match the one-hot vectors representing the words. To resolve this issue, so that one-hot vectors are found by gradient descent, which can be interpreted

as word, terms are added to the error function in order to regularize the decoder input vectors towards one-hot vectors. Different ways to do this regularization is tested.

3.3.1 Entropy

One idea for the regularization is to use entropy. Entropy like cross entropy comes from information theory and measures the average amount of information gained by observing an event, given a probability distribution.

$$H(x) = -\sum_i (x_i \log x_i) \tag{3.3}$$

The intuition for using entropy to regularize, is that no information can be gained from a certain event, meaning the entropy for a one-hot vector is zero and the further from a one-hot vector a distribution is the more entropy it will have.
A problem with using entropy for regularization is that it only works properly as long as long as all values are between zero and one. This can be solved by applying the sigmoid function to all elements of the vector before calculating the entropy. Even if the sigmoid function is applied to all elements of the vector and the entropy function is part the error function, the input vectors to the decoder could still have several elements being one or close to one. This can be solved by adding a term to the error function to give an error when the vectors do not sum to one.

$$N(x) = (1 - \sum_i x_i)^2 \tag{3.4}$$

If the sigmoid function is applied to the vectors before calculating the error, the vectors will be regularized towards vectors that give one-hot vectors after the sigmoid function is applied, rather than regularize the actually vectors towards one-hot vectors. Therefore the sigmoid function should be applied to the vectors before feeding them to the decoder as well.

$$E_1(T, \hat{T}) = \sum_i aH(\sigma(t_i), \hat{t}_i) + bH(\sigma(t_i)) + cN(\sigma(t_i)) \tag{3.5}$$

Together with the primary part of the error function, the cross entropy of the decoder input and decoder output, the entire error function is given by Equation 3.5. a, b and c in the equation are weights.
Another way to make sure entropy can properly be calculated is to apply softmax to the vectors. This will make sure that the elements of the vectors are between zero and one and that they sum to one. When softmax is applied the term that gives an error if vectors does not sum to one, N, will not be needed. Similar to when applying the sigmoid function softmax should be applied to the vectors before feeding them to the decoder.

$$E_2(T, \hat{T}) = \sum_i aH(s(t_i), \hat{t}_i) + bH(s(t_i)) \tag{3.6}$$

Equation 3.6 shows the error function when softmax is used as part of it.

3.3.2 No activation function

Another way to do the regularization that is tested, is by using the following equation

$$A(x) = \sum_i (||0.5 - x_i| - 0.5|) \tag{3.7}$$

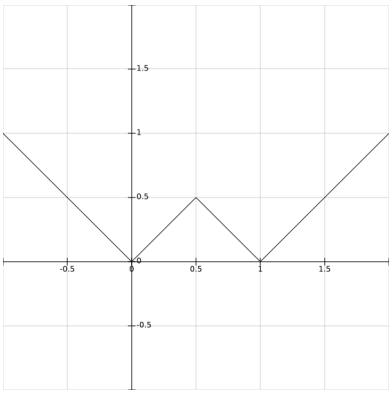

Figure 3.2: Visualizing Equation 3.7 for a x_i in the sum

Figure 3.2 visualizes Equation 3.7 for a x_i in the sum. It can be seen that A is minimized if all x_i in x is set to either zero or one. If a term is added to give an error if x does not sum to one, the function is minimized when all but one element in x is set to zero and one element is set to one.

$$AN(x) = A(x) + N(x) \tag{3.8}$$

If Equation 3.8 is used, x will be regularized towards a one-hot vector and it should not be necessary to apply softmax before feeding the vectors to the decoder.

$$E_3(T, \hat{T}) = \sum_i aH(t_i, \hat{t}_i) + bA(t_i) + cN(t_i) \tag{3.9}$$

Equation 3.9 shows the error function when A and N is used for regularization. Together there are three error functions that are tested in this thesis, E_1, E_2 and E_3. In these functions it is important to set the weights a, b and c so that the regularizing terms are not weighed to much since the important part of the functions is the cross entropy of the decoder input and decoder output.

4
Method

In order to reach the goals of this thesis three separate tests will be performed to test the error functions described in Section 3.3, E_1, E_2 and E_3. First two simpler tests will be performed on some generated data with simpler and smaller models. These test will be useful for developing the error function and to get a fast sens for how well gradient descent decoding might work. After these smaller test a full scale test on the translation model describe in Section 3.1 will be performed.

There are some parameters that needs to be set when using the gradient descent decoding. These are a, b and c in the three error functions, and also the learning rate and learning rate decay for gradient descent. The values for these parameters that give the best results will have to be found and will be presented in the results section.

Another important aspect to the gradient descent decoding is how the input vectors to the decoder are initialized. Usually when training networks the parameters of the network are initialized randomly within a certain interval. An especially interesting way to initialize the vectors would be to initialize them to the probability distributions that are generated when performing a greedy decoding. This should give the gradient descent decoding a good initial start for decreasing the error. A potential problem with initializing the vectors in this way could be that the gradient descent decoding gets stuck in a minimum with the vectors that the greedy decoding finds, and can not improve the probability of the sequence. For the tests that uses an encoder-decoder model four different ways of initializing the decoder inputs will be compared, initializing the elements of the vectors randomly between zero and one, initializing all elements as zero, initializing all elements as one, and initializing the input as the output of a greedy decoding.

4.1 Symbol to symbol test

The symbol to symbol test will test how well gradient descent can be used to find the input of a network given a desired output. The test will be performed on a simple one layer network supposed to translate one symbol, represented by a one-hot vector, into another one. Gradient descent will alter the input in order to minimize the cross entropy of the desired output and the output of the network. The data the network will be taught is the following

input	output
0	3
1	1
2	0
3	2

Table 4.1: Translation table for symbol to symbol test

The numbers in Table 4.1 represent the index of the one in the one-hot vectors. Besides testing if gradient descent can find an input that gives a specific output, the different regularization methods described in Section 3.3 can be tested.

4.2 Sequence to sequence test

The sequence to sequence test will test gradient descent decoding on the same kind of model described in Section 3.1. To keep the test small and simple, the sequences the network will learn will be 5 symbols long and consisting of 8 different symbols. The network was trained to learn the following function

$$f(S)_i = \sum_{j=0}^{i} S_j \mod 8 \tag{4.1}$$

For example giving the network the sequence $[2, 2, 2, 2, 2]$ should give the output $[2, 4, 6, 0, 2]$.
This test will further show how the gradient descent decoding can perform on sequences and how the regularization affects it. In this test gradient descent decoding can also be compared to beam search decoding. The different decoding methods will be compared by testing them on some sequences that has not been used when training the network. The values that will be compared will be the average cross entropy of desired output and the decoder output, the average cross entropy of the decoder input and the decoder output (the main part of the error function), and the percentage of correctly decoded symbols. A symbol is seen as correctly decoded if the vector found by gradient descent has its maximum element on the index that matches the desired symbol in the sequence.

4.3 Translation test

The full scale test will include training the model describe in Section 3.1 on the the europarl French-English corpus. To evaluate the performance of the gradient descent decoding it will be compared to greedy and beam search decoding. To compare the the different decoding methods they will be used to translate sentences from the validation data set. As with the sequence to sequence test the different decoding methods will be compared in three ways. By comparing the average cross entropy of the desired output and the decoder output, the average cross entropy of the decoder input and the decoder output and the BLEU score of the translations.

5
Experimental setup

There are multiple parameters used for the models, this section will present the parameters and, the training and evaluation data used when performing the tests described in Section 4.

5.1 Symbol to symbol test model

The model for the symbol to symbol test was trained for ten epochs with a learning rate of 0.9 and no learning rate decay.

5.2 Sequence to sequence test model

The model used for the sequence to sequence test is the same model described in Section 3.1. It uses an embedding of size 32 and the recurrent part of the network consists of a GRU with a size of 32. The data used consisted of every permutation of the 8 symbols in a sequence of 5 symbols, together with their translations according to Equation 4.1. Of this data 80% was randomly selected and used for training and the rest was used for validation and testing. The model was trained for ten epochs using a learning rate of 0.05 and no learning rate decay.

5.3 Translation model

Training a large artificial neural network can take a lot of time and also require a lot of memory. Therefore parameters for the translation model was used so that the model did not become to large. The vocabularies was built from the 40.000 most common tokens from the training data for English and French separately. This means the word vectors feed to the network had a size of 40.000. The size for the embedding layer of the network was 128 and the size of the GRU cells was 512. Two GRU cells was used together to make up the recurrent part of the network.
The network was trained on the europarl French-English corpus, the training part of the data consisted of 22520400 English and French sentences. The network was trained using a learning rate that started at 0.25 and was halved 6 times during the training that consisted of one epoch. When the training was finished, evaluation was performed on 64 sentences from the validation data set from the corpus.
Because batches was used during training the sequences for a batch needed to be of the same size, to keep the model from being to big this size was set to 15. This

meant that any sentences feed to the network was cut after 15 words if the sentence was longer than 15 words, and padded with a pad symbol if a sentence was shorter than 15 words.

6
Results

This section will present the results from the three tests describe in Section 4 and discuss these results.
When gradient descent was used in the tests the number of iterations the gradient descent was applied was set to 24. This was chosen so that the gradient descent decoding could be performed in a reasonable time.

6.1 Symbol to symbol test

The symbol to symbol test showed that the gradient descent has no problems finding an input that gives a desired output. Without any regularization, trying to find the input that gives the symbol 3 as an output, gave the following input and output for the network:

input:	[	2.34	-0.60	-0.68	-0.52]
output:	[	0.00	0.00	0.00	1.00]

As can be seen the input found by gradient descent is not a proper one-hot vector, but it can quite clearly be seen that the zeroth element is the largest, which would indicate the symbol 0. This is accurately translated by the network to the symbol 3 as the output shows. Decoding the same symbol with the two regularization methods gave the following result:

E_1					
input:	[	0.92	0.03	0.03	0.03]
output:	[	0.04	0.04	0.03	0.88]
E_2					
input:	[	0.99	0.00	0.00	0.00]
output:	[	0.03	0.03	0.02	0.92]
E_3					
input:	[	0.98	-0.12	-0.12	-0.08]
output:	[	0.02	0.02	0.02	0.94]

As can be seen using E_1 and E_2 results in an input a bit closer to a one-hot vector compared to using no regularization. E_3 gives an input that is close to a one-hot vector but its values are not completely within the range zero to one. This is because the sigmoid or softmax function is not applied to the vector as it is for E_1 and E_2.

6.2 Sequence to sequence test

The sequence to sequence test gave some useful insight to how the error function affects the gradient descent decoding. For the particular test data used the gradient descent decoding could perform on par with beam search decoding. Testing showed that using momentum with the gradient descent improved the results, the momentum parameter was set to 0.5. The initialization method that worked best was to initialize the vectors as the output of a greedy decoding, second to that was to initialize the vectors to zero vectors.

decoding method	ce desired	ce decoder	correct symbols
greedy	0.52	0.201	83%
bsd 3	0.57	0.195	84%
bsd 32	0.57	0.195	84%
gdd E_1	0.64	0.517	68%
gdd E_2	0.51	0.199	73%
gdd E_3	1.33	3.453	21%

Table 6.1: Average cross entropy of desired output and decoder output (ce desired), average cross entropy of decoder input and decoder output (ce decoder), and percentage of correctly predicted symbols from different decoding methods, calculated over 64 sequences. Bsd 3 and bsd 32 stands for beam search decoding using a beam width of 3 and 32 respectively. Ggdd $E_1/E_2/E_3$ is gradient descent decoding using E_1, E_2 and E_3 as error functions.

Table 6.2 shows that gradient descent decoding using E_3 as error function gives the worst results. The parameters used with E_3 was a learning rate of 0.01 without any learning rate decay, a, b and c in Equation 3.9 was all set to 1. Since using E_3 did not give any good results, more focus was put on using E_1 and E_2.
When using E_1 and E_2 it was found that a high learning rate could be used, since the sigmoid function and softmax was applied to the vectors after gradient descent was applied, it did not matter if the vectors was changed in a large way. It was also noticed that using no learning rate decay gave better results. Since changes to early vectors in the sequence can change what the optimal vectors are later in the sequence it is beneficial to keep the learning rate high so that large changes to the vectors can still be made by gradient descent by the end of the decoding. To get the result for E_2 in Table 6.1 a learning rate of 0.75 and no learning rate decay was used, a and b in Equation 3.5 was set to 1 and 0 respectively. The results for E_1 in table 6.1 was given using a learning rate of 1 and no learning rate decay, a, b and c was set to 1, 0 and 50 respectively. For all variants of gradient descent decoding in Table 6.1 the input vectors to the decoder was initialized as zero vectors.
Two problems were found when using E_2 as error function. One was that even if the decoder was feed something quite close to a one-hot vector it might not predict the next symbol as it would, had it gotten a one-hot vector, this meant that the gradient descent optimized for wrongly predicted vectors later in the sequence. The other problem was the gradient descent could fall into a local minimum. Early on during the decoding some symbols might not be accurately predicted yet because

earlier vectors in the sequence have not been properly optimized yet. This can mean that gradient descent starts optimizing later vectors in the sequence after wrongly predicted symbols, later the prediction for those symbols might change to a correct prediction, by this time gradient descent might have formed strong one-hot vectors and can not change the vectors without increasing the error first.

An example of the first problem is when tying to translate the sequence $[2, 2, 2, 2, 2]$, this results in the sequence $[2, 4, 6, 6, 2]$ after $argmax$ is applied to the vectors found by gradient descent. At the same time, the sequence predicted by the decoder is $[2, 4, 6, 6, 2]$ so the decoded sequence matches the predicted one. When the sequence $[2, 2, 2, 2, 2]$ is translated using a greedy decoder the predicted sequence was $[2, 4, 6, 0, 2]$. As can be seen using the gradient descent decoding predicts the wrong fourth symbol in the sequence even though it has been feed the same symbols as the greedy decoder, up to that point. The problem is that the third vector in the sequence the gradient descent finds and feeds back to decoder is approximately $[0.43, 0.00, 0.02, 0.00, 0.04, 0.00, 0.50, 0.00]$. Feeding that vector does not lead to the same prediction that feeding the proper one-hot vector $[0, 0, 0, 0, 0, 0, 1, 0]$ does. To solve this the regularization term in E_1 could be weighted more i.e., set b to a larger value.

An example of the second problem is translating $[0, 7, 2, 4, 6]$. This is decoded to $[0, 7, 1, 5, 1]$ while the network predicts $[0, 7, 1, 5, 3]$, which is the correct sequence. The decoder is supposed to be able to find symbols that does not match the prediction as it can lead to other prediction later in the sequence and a more probable sequence can be found. In this case its only the last symbol that diverge from the prediction. Since the last symbol feed to the network does not affect the rest of the sequence in any way it should match the prediction to minimize the error. The problem is that in an earlier stage of the decoding 1 is predicted as the last symbol since earlier symbols has not been optimized properly yet, after the first time gradient descent is applied the decoded sequence is $[0, 7, 1, 1, 1]$ and the predicted sequence $[0, 7, 1, 1, 1]$. When later in the decoding process the right symbol is predicted gradient descent does not change the last symbol accordingly. In this case applying gradient descent for a couple more iterations could solve the problem, but in some cases especially if b was non zero, gradient descent did not seem to be able to change some symbols after the first couple of iterations. The best results were given when b was set to zero so that entropy gave no error.

Some sequences could get better decoded using gradient descent decoding compared to greedy decoding, for example the sequence $[3, 0, 1, 0, 5]$. Using gradient descent decoding, the sequence $[3, 0, 1, 0, 5]$ was decoded into $[3, 3, 4, 4, 1]$ as it should be, where as using greedy decoding results in the sequence $[3, 3, 4, 0, 3]$. In this case it is probably just coincidence as the sequence $[3, 3, 4, 0, 3]$ is more probable according to the network and if gradient descent decoding worked properly it should have given the result $[3, 3, 4, 0, 3]$. The reason it does not is probably because of the problems described above.

The term in E_1, Equation 3.5, meant to give an error if the vectors did not sum to one had to be set to a very high value for the vectors to actually sum to one. The two problems when using E_2 describe where also noticed when using $E1$.

decoding method	ce desired	ce decoder	correct symbols
greedy	0.52	0.201	83%
bsd 3	0.57	0.195	84%
gdd E_1	0.52	0.222	83%
gdd E_2	0.51	0.199	84%

Table 6.2: Average cross entropy of desired output and decoder output (ce desired), average cross entropy of decoder input and decoder output (ce decoder), and percentage of correctly predicted symbols from different decoding methods, calculated over 64 sequences.Bsd 3 stands for beam search decoding using a beam width of 3. gdd E_1/E_2 is gradient descent decoding using E_1 and E_2 as error functions.

When initializing the vectors as zero vectors the results did not perform very well compared to greedy and beam search decoding, as can be seen in Table 6.1 . However when initializing the vectors as the output of a greedy decoding the results where better, shown in Table 6.2. The result for E_2 in Table 6.2 was given using a learning rate of 1 with no learning rate decay, a and b was set to 1 and 0 respectively. The results for E_1 in Table 6.2 was given using a learning rate of 2 and no learning rate decay, a, b and c was set to 1, 0 and 500 respectively. Using E_2 reduces the negative log probability of the found symbols and predicted symbols and accurately decoded more symbols compared to greedy decoding. Using E_1 increases the cross entropy of the desired output and the decoder output and the cross entropy of the decoder input and decoder output, compared to greedy decoding even tough it was initialized using the distributions generated by a greedy decoding. So E_1 does not seem to work very well. However using E_2 reduces both cross entropy values and increases the number of correctly decoded symbols. This said the small differences in the results are not really enough to make any definite conclusions.
Something to note is that beam search decoding does not give better results when using a beam width of 32 compared to a beam width of 3, which in turn is barley better than greedy decoding. This match findings in [13] where they find that their system does well with a beam width of one (greedy decoding) and that using a beam width of two provides most of the benefits of beam search. This means the beam search and greedy decoding is most probably very close to having an optimal cross entropy of decoder input and decoder output. This means that gradient descent decoding can not be expected to give much better results.

6.3 Translation test

The full scale translation test gave similar results to the sequence to sequence test. The same weights and learning rates were used for gradient descent decoding with E_2 as in the sequence to sequence test i.e. learning rate set to 1, no learning rate decay, and, a and b set to 1 and 0, also momentum was used with the parameter set to 0.5. Since using using E_1 and E_3 did not give as good results as E_2, only the results from E_2 will be displayed and discussed.

decoding method	ce desired	ce decoder	BLEU score
greedy	5.67	0.68	11.7
bsd 3	5.61	0.62	10.5
gdd E_2	6.26	0.69	11.9

Table 6.3: Average cross entropy of desired output and decoder output (ce desired), average cross entropy of decoder input and decoder output (ce decoder), and BLEU score from different decoding methods, calculated over a 64 sentences. Bsd 3 stands for beam search decoding using a beam width of 3 and gdd E_2 is gradient descent decoding using E_2 as error function.

As can be seen in Table 6.3 the results of both gradient descent decoding and beam search decoding does not give any significant different results compared to greedy decoding. Noticeable is how beam search decoding reduces the average cross entropy of the decoder input and decoder outputs, as it is supposed to do, but decreases the BLEU score for the translated sentences. Also gradient descent decoding using E_2 as error function increases the cross entropy but increases the BLEU score. Most likely this is just coincidence and bigger differences in the result would be desirable to properly compare the methods. BLEU score works better if there are several reference translation to compare the generated translation with but for the test data used, there where only one reference translation per sentence. Beam search decoding would perhaps give a relatively better BLEU score if more reference translations could be used.

input sentence	And that's not all.
reference	Bien plus.
decoding method	translation
greedy	Et cela ne s'est pas complètement.
bsd 3	Et ce n'est pas tout ça.
gdd E_2	Et cela ne s'est pas tout.

Table 6.4: The table shows how the different decoding methods translates the sentence "And thats not all"

For the most part the gradient descent decoding did not change the translations compared to greedy decoding, which it was initialized to. When it did change something in the translation, the difference was mostly only one word changed to some other synonym or similar word. Table 6.4 shows a case of this, the gradient descent decoding has changed the word "complètement" to "tout" from the greedy decoding. The beam search decoding can give translations that can differ a bit more as can be seen in Table 6.4. In this case none of the translations got a very good BLEU score since the reference translation is "Bien plus.".
If the decoder inputs were initialized to zero vectors gradient descent decoding does not work well at all and seems to give pretty much random sequences of tokens. This together with the fact that the sentences does not change much when the decoder input is initialized as the output of a greedy decoding, suggests that the gradient descent decoding does not work very well for the full translation model. This may

be because the vectors that the gradient descent tries to adjust are much larger than the ones for the sequence to sequence test. Together with the large size of the network the calculations for the gradients with the backpropagation might be to complex for gradient descent to minimize the error function effectively.

6.4 Efficiency

Another interesting part of how well gradient descent decoding performs is how fast it can decode translations compared to beam search and greedy decoding. Unfortunately the implementations of both beam search and gradient descent decoding were not very good in this respect. Both gradient descent and beam search decoding were several times slower than the greedy decoding.
If C is the complexity for the decoder generating a probability distribution given an input to the decoder, k is the beam width, n is the length of the sentence, and m is the number of tokens the network knows, the complexity for beam search decoding should be

$$O(k * n * m * \log(k * m) * C) \tag{6.1}$$

With i being the number of iteration that gradient descent is applied, the complexity of of gradient descent decoding should be

$$O(i * n * C) \tag{6.2}$$

Seemingly the beam width does not need to be much larger than two in order to gain most of the benefits of beam search. This means that gradient descent has to find a good translations in a number of iterations that is not much bigger than $k * m * \log(k * m)$ in order to be a good substitute for beam search.

7
Future work

Looking at the results of this thesis there are a lot of future work that could be done. First of all the results of comparing the gradient descent with greedy and beam search decoding, showed no major differences between the decoding methods. It could therefore be interesting to perform more tests, similar to the ones in this thesis, using more data. This could allow for more decisive conclusions to be made. In this thesis the gradient descent decoding was only tested with a basic gradient descent optimization algorithm with the addition of momentum. There are many different gradient descent optimization algorithms that could be used. With a better optimization algorithm better translations could probably be found faster.
Future work could also be done by implementing beam search and gradient descent decoding so they are more efficient. This would allow for the speed of the two decoding methods being properly comparable. Furthermore, in this thesis the number of iterations that gradient descent was performed during the decoding was set to be 24. It would be interesting to know how large this value would need to be in order for gradient descent to converge and how good the decoding would be at that point. Also how few iterations could be performed while still decoding well would be interesting to test.

8

Conclusion

In this thesis decoding neural machine translation by using gradient descent have been tested. Three different error functions to be used with the gradient descent where developed, tested and compared to greedy and beam search decoding. Out of the three error functions the one that work best used the cross entropy of the decoder input and the decoder output and applied softmax to the decoder input. Using this error function, decoding using gradient descent could give similar results to beam search decoding. When used to decode small generated test data the gradient descent decoding worked well, but for the large translation model gradient descent decoding did not seem to work very well. In the tests performed, beam search did not give a significant improvement over greedy decoding, which means that gradient descent decoding perhaps can not be expected to give much improvement either. More test data should be used to be able to better compare beam search and gradient descent decoding, but with some more work put into improving gradient descent it could have some potential.

Bibliography

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

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

[3] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. A neural algorithm of artistic style. *arXiv preprint arXiv:1508.06576*, 2015.

[4] Alex Graves, Marcus Liwicki, Santiago Fernández, Roman Bertolami, Horst Bunke, and Jürgen Schmidhuber. A novel connectionist system for unconstrained handwriting recognition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 31(5):855–868, 2009.

[5] Philipp Koehn. Europarl: A parallel corpus for statistical machine translation. In *MT summit*, volume 5, pages 79–86, 2005.

[6] Philipp Koehn, Franz Josef Och, and Daniel Marcu. Statistical phrase-based translation. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pages 48–54. Association for Computational Linguistics, 2003.

[7] Minh-Thang Luong, Ilya Sutskever, Quoc V Le, Oriol Vinyals, and Wojciech Zaremba. Addressing the rare word problem in neural machine translation. *arXiv preprint arXiv:1410.8206*, 2014.

[8] T. Mikolov. *Statistical Language Models based on Neural Networks*. PhD thesis, Brno University of Technology, 2012.

[9] 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, 2002.

[10] Henry A Rowley, Shumeet Baluja, and Takeo Kanade. Neural network-based face detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 20(1):23–38, 1998.

[11] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

[12] Hasim Sak, Andrew W Senior, and Françoise Beaufays. Long short-term memory recurrent neural network architectures for large scale acoustic modeling. In *INTERSPEECH*, pages 338–342, 2014.

[13] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.