FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Rozpoznávání souvislé řeči s využitím neuronových sítí
Student:	Adam Zvada
Vedoucí:	Ing. Miroslav Skrbek, Ph.D.
Studijní program:	Informatika
Studijní obor:	Teoretická informatika
Katedra:	Katedra teoretické informatiky
Platnost zadání:	Do konce letního semestru 2018/19

Pokyny pro vypracování

Provedte rešerši metod pro rozpoznávání souvislé řeči s využitím neuronových sítí. Uvažujte rekurentní neuronové sítě a zvažte také možnost použití neuronových turingových strojů. Na základě rešerše a po dohodě s vedoucím práce vyberte vhodné řešení pro robota NAO. Maximálně využívejte existujících knihoven s implementacemi potřebných metod. Navržené řešení otestujte na reálných datech. Rozsah práce upřesněte po dohodě s vedoucím práce.

Seznam odborné literatury

Dodá vedoucí práce.

doc. Ing. Jan Janoušek, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 13. února 2018

Czech Technical University in Prague
Faculty of Information Technology
Department of Theoretical Informatics

Bachelor's thesis

Continuous Speech Recognition by Neural Networks

Adam Zvada

Supervisor: Ing. Miroslav Skrbek Ph.D

April 26, 2018

Acknowledgements

THANKS

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as school work under the provisions of Article 60(1) of the Act.

In Prague on April 26, 2018 .....

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Zvada, Adam. *Continuous Speech Recognition by Neural Networks.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2018.

Abstrakt

V několika větách shrňte obsah a přínos této práce v českém jazyce.

Klíčová slova Replace with comma-separated list of keywords in Czech.

Abstract

In my bachelor thesis Summarize the contents and contribution of your work in a few sentences in English language.

Keywords Replace with comma-separated list of keywords in English.

Contents

List of Figures

Introduction

Chapter 1

Introduction to Speech Recognition

Introduction to Speech Recognition

Chapter 2

Neural Network

Neural networks have remarkable ability to derive meaning from complicated data. They can be used to extract patterns and detect trends that are too complex to be noticed by either humans or other computer techniques. Even though they have been around since the 1950s, it is only in the last decade when they started to outperform robust system or even humans in specify tasks. However, they require a huge amount of training examples and computational power to be trained for preforming a reasonable prediction. Fortunately, GPUs has seen enormous increase in performance[1] and 90% of the data in the world today has been created in the last two years alone, at 2.5 quintillion bytes of data a day*Refrence data*. That's why ANN is big topic in Computer Science and in the technology industry and it currently provides the best solutions to many problems such as speech recognition, image recognition, and natural language processing.

2.1 Inspiration in Nature

Artificial neural network (ANN) is heavily inspired by the way how biological neural networks process information in the human brain. Even though our brain is extremely complex and still not fully understand, we just need to know how information is being transferred. The basic building block is nerve cell called *neuron*. It receives, processes, and transmits information through electrical and chemical signals. It's estimated that an average human has 86 billion neurons *.

Dendrites are extensions of a nerve cell that propagate the electrochemical stimulation received from other neurons to the cell body. You may think of them as inputs to neuron, whereas neuron's output is called *axon*, a long nerve fiber that conducts electrical impulses away from the cell body. The end of axon is branched to many axon terminals which can be again connected to

[1] WHY GPU FRO DEEP LEARNING.

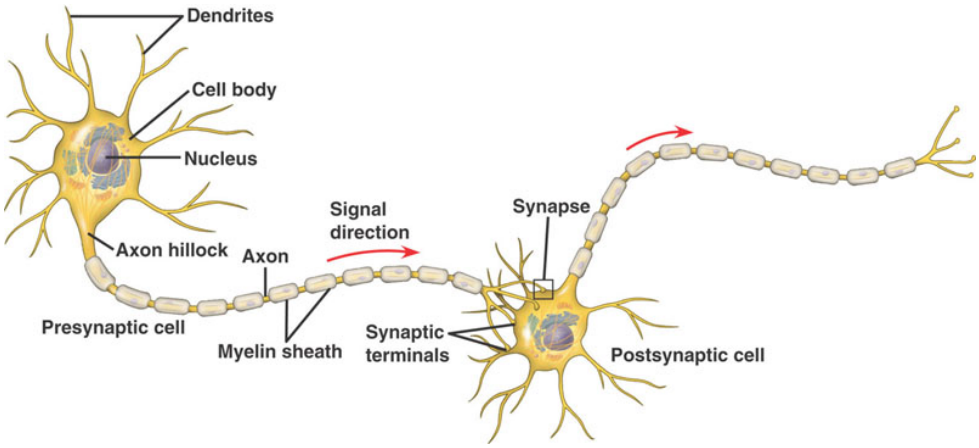

Figure 2.1: Illustration of nerve cell and communication flow

other dendrites. The connection is managed by *synapses* that can permit the passing of electrical signal to cell body. Once the cell reaches a certain threshold, an action potential will fire, sending the electrical signal down the axon to other connected neurons.

2.2 Artificial Neuron

Artificial neuron is a generic comuptational unit, basic building block for artificial neural network (ANN). It's simplified version of the biological counterpart and we are able to map parts of biological neuron with the artificial one. It takes n inputs represented as a vector $x \in \mathbb{R}^n$ which correspond to dendrites. Generaly artificial neuron produces single output $y \in \mathbb{R}$ as biological neuron where we call it axon. Each neuron's input $i = 1, 2, \ldots, n$ has assigned weight (synapse) $w_1, w_2 \ldots w_n$, they refer to the connection strength between neurons. Weights and same as for synapse are the backbone of learning because in training phases, they keep changing to produce wanted output.(*In this chapter, we will elaborate further.) Inside the artificial neuron, input vector with their weights are combined and run through an activation function producing some output y. This process is illustrated in $LINK_PRECEPRTON$.

2.3 Perceptron

Perceptron is the simplest ANN with just one neuron and since we covered the basic intuition about artificial neuron we may proceed further and take a look at how output is actually calculated. The equation for a perceptron can be written as

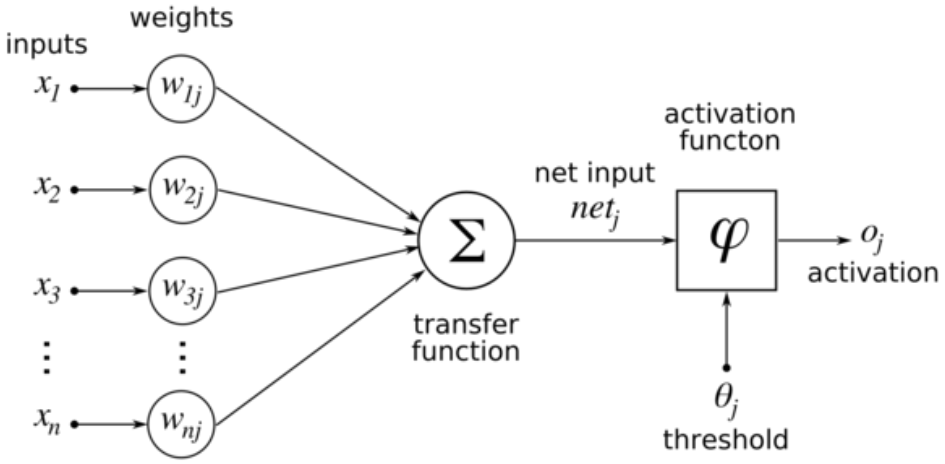

Figure 2.2: Illustration of nerve cell and communication flow

$$y = f(\sum_{i=1}^{N} w_i \cdot x_i + b)$$

where

- x - input vector
- y - predicted output
- f - activation function
- w - weights
- b - bias

Perceptron is a basically linear classifier, therefore the data has to be linearly separable otherwise we would not be able to make the correct prediction. Problems such as speech recognition are not definitely linearly separable, however we can solve non-linear decisions for example by introducing another layer of neurons, thus creating *Multilayered Perceptron*.

2.3.1 Activation Functions

We have stated that biological neuron fires electrical signal to other connected neurons whenever it reaches a certain threshold of incoming electrical impulses. Activation function is based on that concept and inside an artificial neuron it is used for calculating output signal via equation*. It introduces non-linear properties to our ANN and without an activation function would be just a regular linear regression model. Nowadays many different activation

function are being used and their performance varies from model to model.

List of some activation function:

- *Sigmoid*
$$\sigma(x) = \frac{1}{1 + e^{-x}}$$
- *Hyperbolic Tangent*
$$\tanh(x) = \frac{(e^{x} - e^{-x})}{(e^{x} + e^{-x})}$$
- *ReLU*
$$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$$
- *Softmax*
$$f_i(\vec{x}) = \frac{e^{x_i}}{\sum_{j=1}^{J} e^{x_j}}, \quad i = 1, 2 \dots J$$
where i is number of output

* TODO: Activation Functions features... differentiable and non-linear*

2.3.2 Bias

We can think of bias as a value stored inside neuron and being used to calculate it's output. The bias value allows the activation function to be shifted to the left or right, to better fit the data.

2.4 Topology of Artificial Neuron Network

Basic ANN as feedforward model is a directed graph with nodes as neurons and edges with weights representing connection to other neurons. ANN can be divided to three important layers as shown in Fig*. Yellow nodes is an input layer which takes input data, dimension of input vector has to correspond to number of input nodes. Hidden layer as the green nodes is most important to ANN and that is where the training and evaluation happens. Number of hidden layers and neurons needs to be in a good ratio between its size and its effectiveness. Output layer produces output vector as the prediction for given input.

2.5 Network Evaluation

Evaluation of ANN is done layer by layer.

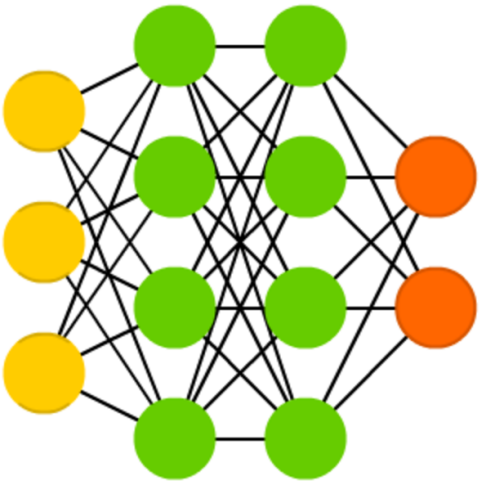

Figure 2.3: Basic topology of fully connected artificial neuron network with input vector of size 3, output vector of size 2 and two hidden layers.

2.6 Training

The greatest trait of ANN is ability to learn from given data and then make the best approximate prediction. The aim of the learning process is to find the most optimal values for network's weights and biases while minimizing error on predicated values. For ANN to learn we have to introduce training data consisted of input vector which will be feeded to the network and desired output value (label) for calculating our loss. This approach is called supervised learning[2].

2.6.1 Loss Function

Loss function compares the prediction from ANN with the desired output and returns the error of the prediction. During a training ANN, the goal is to minimize given loss function. The most common and most intuitive loss function is Mean squared Error (MSE),

$$\mathrm{MSE}(y, \hat{y}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2.$$

2.6.2 Backpropagation

Backpropagation algorithm is responsible for the ability to learn from given training data. It is an iterative algorithm which for each training data from

[2]ANN can be also trained using unsupervised learning.

given training dataset backpropagates the error and adjust the weights and biases accordingly to get desired output.

2.6.2.1 Optimization

Backpropagation requires optimizer to minimize the error on the training data. We will describe backpropagation with using *gradient descent* as the most common optimization algorithm.
Weights and biases are updated using formula,

$$W_{jk}^{l} := W_{jk}^{l} - \alpha \frac{\partial E}{\partial W_{jk}^{l}}$$

$$b_{j}^{l} := b_{j}^{l} - \alpha \frac{\partial E}{\partial b_{j}^{l}}$$

where $\boldsymbol{W_{jk}^{l}}$ is weight with connection between unit j in layer l and unit i in layer $l+1$, $\boldsymbol{b_{j}^{l}}$ is bias associated with unit i in layer $l+1$, $\boldsymbol{\alpha}$ is a learning rate *Reference*, and $\frac{\boldsymbol{\partial E}}{\boldsymbol{\partial W_{jk}^{l}}}$ or $\frac{\boldsymbol{\partial E}}{\boldsymbol{\partial b_{j}^{l}}}$ can be interpreted as minimizing loss function with respect to given weight and bias respectively.

By applying a chain rule twice on the partial derivative of the loss function with respect to a weight, we get

$$\frac{\partial E}{\partial W_{jk}^{l}} = \frac{\partial E}{\partial a_{j}^{l}} \frac{\partial a_{j}^{l}}{\partial z_{j}^{l}} \frac{\partial z_{j}^{l}}{\partial W_{jk}^{l}}$$

where $\boldsymbol{z_{j}^{l}}$ is a sum of weighted inputs to unit j in layer l

$$z_{j}^{l} = b_{j}^{l} + \sum_{k=1}^{K} w_{jk}^{l} a_{k}^{l-1}$$

and $\boldsymbol{a_{j}^{l}}$ is an output of node j in layer l

$$a_{j}^{l} = f(z_{j}^{l}).$$

Let's calculate the last two products of equation *Reference*:

$$\frac{\partial a_{j}^{l}}{\partial z_{j}^{l}} = f'(z_{j}^{l})$$

$$\frac{\partial z_{j}^{l}}{\partial W_{jk}^{l}} = \frac{\partial W_{jk}^{l} a_{k}^{l}}{\partial W_{j} k^{l}} = a_{k}^{l-1}$$

We introduce a new varibale $\boldsymbol{\delta_j^l}$ which represents the error in unit j in layer l and helps us to better understand and calculate real interested value of $\frac{\partial E}{\partial W_{jk}^l}$ and $\frac{\partial E}{\partial b_j^l}$.

$$\delta_j^l = \frac{\partial E}{\partial z_j^l}$$

We will simplify the error equation on neuron j in output layer L as

$$\delta_j^L = \frac{\partial E}{\partial z_j^L} = \frac{\partial E}{\partial a_j^L}\frac{\partial a_j^L}{\partial z_j^L} = \frac{\partial E}{\partial a_j^L} f'(z_j^L)$$

Now we have enough information to reformulate equation *reference* for output layer to

$$\frac{\partial E}{\partial W_{jk}^l} = \delta_k^L a_j^L.$$

However, to be able to update weights inside the hidden layers, we have to redefine the calculation of δ_j^l. We know that the error produced by an output neuron is just influencing the output value but inside a hidden layer the produced error propagates to all following layers. Therefore we have calculate the δ_j^l where layer l is inside a hidden layer and take into account all δ^{l+1} from following layer $l+1$.

$$\delta_j^l = \frac{\partial E}{\partial z_j^l} = \sum_i \frac{\partial E}{\partial z_i^{l+1}}\frac{\partial z_i^{l+1}}{\partial z_j^l} = \sum_i \frac{\partial E}{\partial z_i^{l+1}}\frac{\partial z_i^{l+1}}{\partial a_j^l}\frac{\partial a_j^l}{\partial z_j^l}$$

$$= \sum_i \delta_i^{l+1} W_{ij}^{l+1} f'(z_j^l)$$

where the sum index i iterates over all neurons in layer $l+1$ and Notice that we have substituted $\frac{\partial E}{\partial z_i^{l+1}}$ with δ_i^{l+1} which is calculated from previous iteration. Finally, we may calculate all weights adjustments through the whole network as

$$W_{jk}^l := W_{jk}^l - \alpha \delta_k^l a_j^l$$

where

$$\delta_k^l = \frac{\partial E}{\partial a_j^L} f'(z_j^L), \quad l = L$$

or

$$\delta_k^l = \sum_i \delta_i^{l+1} W_{ij}^{l+1} f'(z_j^l), \quad l = 2, \ldots, L-1.$$

We won't be exampling the equation for biases adjustments because it follows a similar process shown above with just little changes, resulting to equation

$$b_j^l := b_j^l - \alpha \delta_l^j$$

2.6.2.2 Backpropagation Algortihm

Algorithm 1 Backpropagation

1: Initialize network weights and biases
2: **for each** training data from training dataset **do**
3: Forward pass and calculate network prediction for given training input
4: Calculate error δ^L for output layer
5: Calculate errors δ^l for hidden layers
6: Update weights and biases using precalculated δ^l

Chapter 3

Recurrent Neural Network

Neural networks are powerful learning models that achieve state-of-the-art results in a wide range of machine learning tasks. Nevertheless, they have limitations in the field of sequential data. Standard ANNs rely on the assumption of independence among the training examples but if data points are related in time or space then ANNs would not be the right model for the task*Reference - arvix*.
Recurrent neural network (RNN) is type of neural network which is precisely designed to work with sequential data through time. The key difference is that RNN's neurons in hidden layer have a special edge (recurrent edge) to a next time step which can be interpreted as a loop. In RNN, the neuron's output is dependent on the previous computations which is sent through the recurrent edge. Basically, the recurrent edges or loops allow persistence of information from one time step to the next one as shown on *Figure 3.1*.

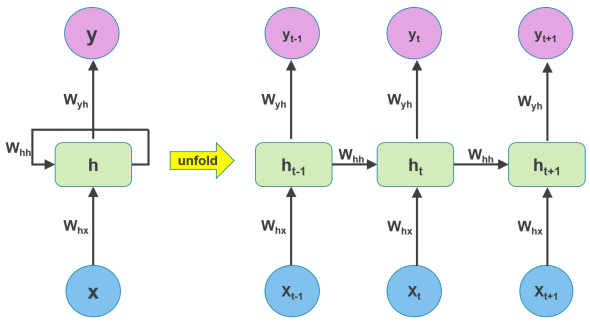

Figure 3.1: Simple RNN topology and illustration of unrolled RNN through time

3.1 Evaluation

In *Figure 3.1.* we may see simplification of evaluation process of RNN through the time steps. RNN's neuron cell in hidden layer takes two inputs, x_t and h_{t-1} which is value (hidden state) sent through the recurrent edge from previous time-step. The cell also produces two outputs, h_t as hidden state for upcoming time-setp

$$h_t = f(W_{hx}x_t + W_{hh}h_{t-1} + b_h)$$

where f is arbitrary non-linear activation function, W_{hx} is matrix of conventional weights, W_{hh} is the matrix of recurrent weights and b_h is a bais. The second output from cell is y_t which outputs the predication using precalculated hidden state h_t,

$$y_t = W_{hy}h_t + b_y$$

where W_{hy} is matrix of output weights.

3.1.1 Softmax Fucntion

It is very common for RNN models to use *softmax* as activation function for output layer. Softmax function helps to get probability distribution of outputs so it's useful for finding most probable occurrence of output with respect to other outputs.

$$\text{softmax}(y)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}, \quad \text{for } j = 1, \ldots, K$$

Softmax is being used for calculting output value of y_t resulting to formula

$$y_t = \text{softmax}(W_{hy}h_t + b_y).$$

3.2 Training

Training a RNN is similar to training a traditional ANN. We also use the backpropagation algorithm, but since the parameters are shared by all time-steps in the network, the gradient at each output depends not only on the calculations of the current time-step, but also the previous time-steps.

3.2.1 Backpropagation Through Time

The most used algortihm to train RNN is *backpropagation through time* (BPTT), introduced by Werbos in 1990. BPTT is basically an extanded version of back-propagation algortihm where we not only propagate the error to all following layers but also through the hidden states. We may think of it as unrolling the RNN to sequence of identical ANNs where the recurrent edge connects

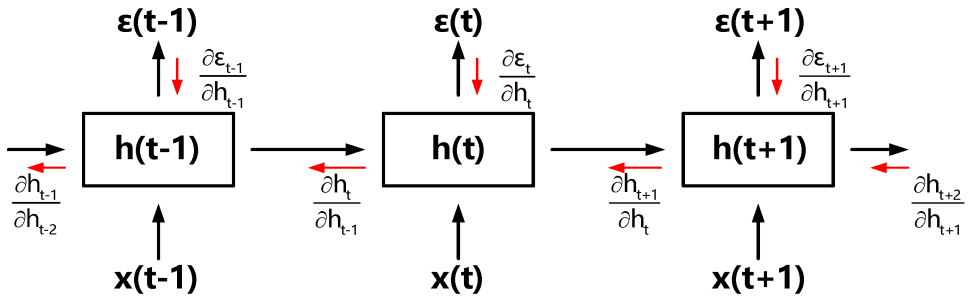

Figure 3.2: Deriving the gradients according to the back-propagation through time (BPTT) method. Notaion for output value $\epsilon(t)$ corresponds to our y_t.

the sequences of neurons in hidden layer together as shown on *Figure 3.1 and 3.2*. On *Figure 3.2* is also indicated how the errors are propagated. The propagation of errors through hidden states allows the RNN to learn long term time dependencies. The calculated gradients of the loss function for defined parameter (W, b) through the sequence of unrolled RNN are then sum up, producing the final gradient for updating the weights or baises, *Equation bottom*.

$$\frac{\partial E}{\partial W_i j^l} = \sum_{t=1}^{T} \frac{\partial E_t}{\partial W_i j^l}$$

where E is predefined loss function, W_{jk}^l is weight with connection between unit j in layer l and unit i in layer $l+1$, T is number of input sequences and $\frac{\partial E_t}{\partial W_{ij}^l}$ is calculated similarly as in backpropagation with just considering existance of recurrent edges

$$\frac{\partial E_t}{\partial W_{ij}^l} = \sum_{k=1}^{t} \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W_{ij}^l}$$

To compute the $\frac{\partial h_t}{\partial h_k}$ we use simple chain rule over all hidden states in interval [k,t].

$$\frac{\partial h_t}{\partial h_k} = \prod_{i=k+1}^{t} \frac{\partial h_i}{\partial h_{i-1}}$$

Putting equations together, we have the following relationship.

$$\frac{\partial E}{\partial W_{ij}^l} = \sum_{t=1}^{T} \sum_{j=1}^{t} \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \left(\prod_{i=k+1}^{t} \frac{\partial h_i}{\partial h_{i-1}} \right) \frac{\partial h_k}{\partial W_{ij}^l}$$

3.2.2 Exploding and Vanishing Gradients

Even though, RNNs had achieved success in learning short-range dependencies, they haven't been showing any worth mentioning achievement with learn-

ing mid-range dependencies. That was mainly cause by problems of *vanishing* and *exploding gradients*, introduced in Bengio et al. (1994).

The exploding gradient problem occurs when backpropagating the error across many time steps, that could lead to exponetilly grow of gradient for long-term components. Basically, a small change in parameters at initial stages can get acumulatd through the time-steps resulting to the exponetilly grow. The values of weights can become so large as to overflow and result in *NaN* values.

The vanishing gradient problem refres to oposite behavior when the gradient values are shrinking exponentially fast and eventually vanishing completely. Gradient contributions from later time-steps become zero and the states at those steps doesn't contribute so we end up not learning long-range dependencies. Vanishing gradients aren't exclusive to RNNs, they also happen in deep ANN.

3.2.2.1 Solutions

To overcome problem with exploding gradient we can *gradient cliping* method. To fix the problem of vanishing gradient is little more complicated. We can avoid it by initilizing the weights carufuly.

3.3 LSTM

Long-Short-Term-Memories (LSTM) is special kind of RNN cell, introduced by Hochreiter Schmidhuber in 1997 *REFERENCE*. Conventional RNNs are only just able to learn short-term dependencies because of vanishing gradient problem. However, LSTM are not effected by it and are capable of learning long-term dependencies. However, LSTM are not effected by it and are capable of learning long-term dependencies.

RNN are not used anymore, just LSTM...

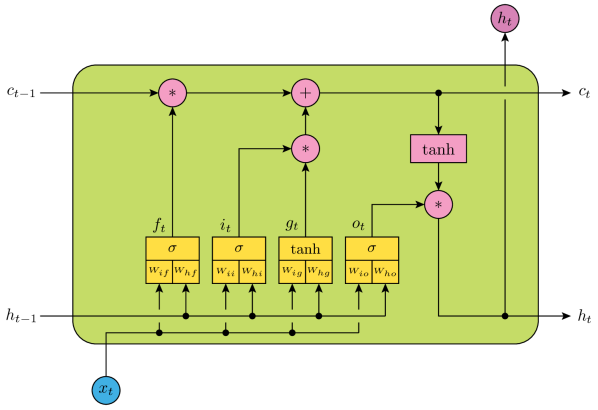

Figure 3.3: Diagram of LSTM cell.

3.4 CTC

Conceptually, BPTT works by unrolling all input timesteps. Each timestep has one input timestep, one copy of the network, and one output. Errors are then calculated and accumulated for each timestep. The network is rolled back up and the weights are updated.

Even though these networks had achieved success in learning short-range dependencies, they haven't been showing any worth mentioning achievement with learning mid-range dependencies. This was mainly caused by the problems of vanishing and exploding gradients.http://arxiv.org/abs/1506.00019

Chapter 4

Implementation

Implementation

Chapter 5

Experiments

Experiments

Conclusion

Appendix A

Acronyms

GUI Graphical user interface

XML Extensible markup language

Appendix B

Contents of enclosed CD

- `readme.txt` ..... the file with CD contents description
- `exe` ..... the directory with executables
- `src` ..... the directory of source codes
  - `wbdcm` ..... implementation sources
  - `thesis` ..... the directory of LaTeX source codes of the thesis
- `text` ..... the thesis text directory
  - `thesis.pdf` ..... the thesis text in PDF format
  - `thesis.ps` ..... the thesis text in PS format