Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Marina Knabbe

Titel

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Marina Knabbe

Titel

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor of Science Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. ErstprÃ¼fer
Zweitgutachter: Prof. Dr. ZweitprÃ¼fer

Eingereicht am: 1. Januar 2345

Marina Knabbe

Thema der Arbeit
Titel

Stichworte
SchlÃ¼sselwort 1, SchlÃ¼sselwort 2

Kurzzusammenfassung
Dieses Dokument …
Marina Knabbe

Title of the paper
English title

Keywords
keyword 1, keyword 2

Abstract
This document …

Inhaltsverzeichnis

Abbildungsverzeichnis

Listings

1 Einleitung (unfinished)

...

2 Künstliche neuronale Netze (unfinished)

- Ablauf: Training, Testing, Using?
- künstliche Intelligenz und Maschinen lernen
- Was können sie: label und prediction

http://deeplearning4j.org/neuralnet-overview

Neural networks are a set of algorithms, modeled loosely after the human brain, that are designed to recognize patterns. They interpret sensory data through a kind of machine perception, labeling or clustering raw input. The patterns they recognize are numerical, contained in vectors, into which all real-world data, be it images, sound, text or time series, must be translated. They help group unlabeled data according by similarities among the example inputs, and they classify data when they have a labeled dataset to train on. To be more precise, neural networks extract features that are fed to other algorithms for clustering and classification.

2.1 Feedforward Netzwerke (unfinished)

Ein Feedforward Netzwerk besteht aus Layern und Neuronen. Die Neuronen sind für die Berechnungen zuständig während die Layer den Aufbau des Netzes bestimmen. Abbildung 2.1 zeigt einen möglichen Aufbau eines künstliches Neurons. Dieses Neuron besteht aus 1 bis

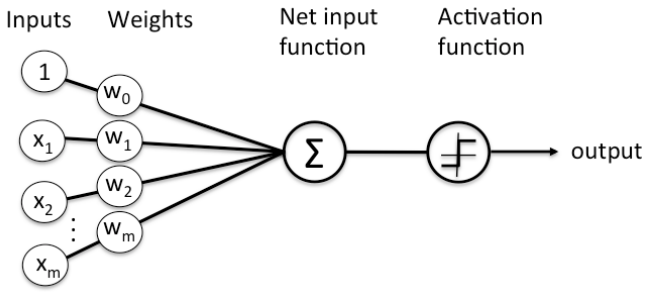


Abb. 2.1: mögliches Aussehen eines Feedforward Neurons (Quelle: [5])

x_m Eingängen (Inputs) mit Gewichten (Weights), einer Input Funktion (Net input function),

einer Aktivierungsfunktion (Activation function) und einem Ausgang (Outputs). Die zu verarbeiteten Daten werden an die Eingänge gelegt, durch die zugehörigen Gewichte verstärkt oder abgeschwächt und anschließend aufsummiert. Die entstandene Summe wird dann an die Aktivierungsfunktion übergeben, welche das Ergebnis dieses Neurons festlegt.

Ein Layer besteht aus einer Reihe von Neuronen beliebiger Anzahl. Ein künstliches neuronales Netz setze sich aus einem Input Layer, einem Output Layer und beliebig vielen Hidden Layern zusammen. Hat ein Netz mehr als ein Hidden Layer so wird es auch als Deep Learning Netz bezeichnet. Abbildung 2.2 zeigt ein Feedforward Netzwerk. Bei diesem Netz besitzt das

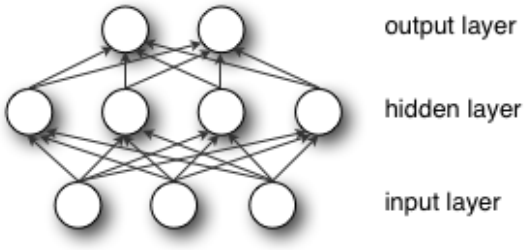


Abb. 2.2: Aufbau eines Feedforward Netzes (Quelle: [4])

Input Layer drei Neuronen, das Hidden Layer hat vier und das Output Layer hat zwei Neuronen. Die Ergebnisse des Input und Hidden Layers dienen dem nachfolgenden Layer als Eingang.

Ein neuronales Netz kann anhand von Trainingsdaten eine Funktion erlernen, indem es die Gewichte verändert. Da bei Trainingsdaten das Ergebnis bekannt ist, muss das Netz die Gewichte so bestimmen, dass sie mit den Eingangsdaten den gewünschten Ausgang mit möglichst kleinem Fehler abbilden. Das Ziel ist möglichst schnell den Punkt zu erreichen an dem der Fehler am kleinsten ist. Um dies zu erreichen wiederholt das Netz die folgenden Schritte: Ergebnis anhand der aktuellen Gewichte bestimmen, Fehler messen, Gewichte aktuallisieren. Eine weitverbreitete Optimierungsfunktions zur Gewichtebestimmung heißt Gradient Descent. Sie beschreibt das Verhältnis des Fehlers zu einem einzelnen Gewicht und wie sich der Fehler verändert wenn das Gewicht angepasst wird.
Sources: - http://deeplearning4j.org/neuralnet-overview

2.2 Recurrent Netzwerke (unfinished)

Recurrent Neuronal Networks (RNN) betrachten im Gegensatz zu Feedforward Netzwerken nicht nur die aktuellen Eingangsdaten sondern auch die vorhergegangenen. Sie besitzen daher zwei Eingangsgrößen, nämlich die gerade angelegten und die zurückgeleiteten aus dem vorherigen Zeitschritt. Abbildung 2.3 zeigt links eine vereinfachte Darstellung eines RNN Neuron mit

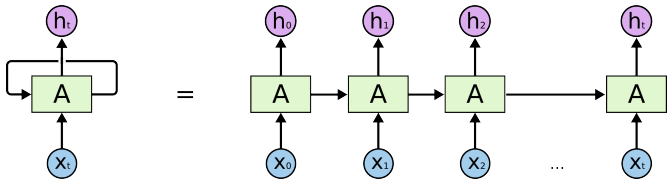


Abb. 2.3: vereinfachte Darstellung eines RNN Neurons (Quelle: [2])

Rückführung, aber ohne Gewichte oder Aktivierungsfunktion. Rechts ist das Ganze als zeitlicher Verlauf dargestellt. Im ersten Zeitschritt wird x_0 an den Eingang gelegt und h_0 als Ergebnis berechnet. Außerdem führt ein Pfeil zum Neuron im zweiten Zeitschritt und dient dort als zweiter Eingang. Das Ergebnis, das ein Neuron liefert ist also immer vom vorherigen abhängig. Man bezeichnet dies auch als Gedächtnis des Netzes. Einem Netz ein Gedächnis zu geben macht immer dann Sinn, wenn die Eingangsdaten eine Sequenz bilden und nicht komplett unabhängig von einander sind. Im Gegensatz zu den Feedforward Netzen können Recurrent Netzwerke Sequenzen erfassen und sie zur Erzeugung ihrer Ausgaben nutzen. Dies ist zum Beispiel bei der automatischen Textgenerierung hilfreich, wo ein folgender Buchstabe immer vom vorherigen abhängt und nicht willkürlich gewählt werden kann. Ein RNN ist in der Lage auf ein q ein u folgen zu lassen um sinnvolle Wörter zu bilden, ein Feedforward Netzwerk kann das nicht. Sources:http://deeplearning4j.org/lstm.html, https://colah.github.io/posts/2015-08-Understanding-LSTMs/

2.2.1 Backpropagation Through Time (unfinished)

Um sinnvolle Ergebnisse von einem Netzwerk zu erhalten müssen die Gewichte solange angepasst werden, bis der Fehler am kleinsten ist. Dies wird mit Hilfe der Backpropagation gemacht, indem Rückwärts vom Fehler über die Ausgänge, die Gewichte und die Eingänge der verschiedenen Layer ein Zusammenhang zwischen Fehlergröße und einzelnen Gewichtseinstellungen hergestellt wird. Durch die Backpropagation lässt sich der Einfluss jedes Gewichtes auf den Fehler ermitteln und mit der Gradient Descent Methode der Fehler verringern. Da bei Recurrent Netzen das Ergebnis und somit der Fehler nicht nur vom aktuellen Zeitschritt abhängt, muss auch die Backpropagation erweitert werden um sinnvoll arbeiten zu können. Backpropagation Through Time (BPTT) ergänzt die normale Backpropagation um den Faktor Zeit, so dass ein Einfluss auf den Fehler von einem Gewicht aus früheren Schritten ermittelt werden kann. Abbildung 2.4 soll diesen Vorgang verdeutlichen. Sie zeigt ein Recurrent Netz das um drei Zeitschritte entrollt wurde indem Komponenten dupliziert wurden. Dadurch lösen sich die Rückführungen auf und das Netzwerk verhält sich wie ein Feedforward Netz. Der

Einfluss jedes Gewichts kann nun anteilig berechnet und anschließend summiert werden, so dass ein einzelner Wert je Gewicht für die Anpassung ermittelt wird. Dieses Verfahren benötigt natürlich mehr Speicher, da alle vorherigen Zustände und Daten für eine bestimme Anzahl an Zeitschritten gespeichert werden müssen.
Sources: http://deeplearning4j.org/lstm.html, 10.1.1.16.6652.pdf

2.2.2 Verschwindende und explodierende Gradienten (unfinished)

Der Gradient stellt die Veränderung aller Gewichte in bezug auf Veränderung im Fehler dar. Wenn der Gradient unbekannt ist, ist eine Veränderung an den Gewichten zur Verkleinerung des Fehlers nicht möglich und das Netz ist nicht in der Lage zu lernen. Zu unbekannten Gradienten kann es kommen, da Informationen die durch ein Netz fließen vielfach multipliziert werden. Multipliziert man einen Betrag regelmäßig mit einem Wert knapp größer Eins kann das Ergebnis unmessbar groß werde und in diesem Fall spricht man von einem explodierenden Gradienten. Umgekehr führt eine widerholte Multiplikation eines Betrages mit einem Wert kleiner als Eins zu einem sehr kleinem Ergebnis. Der Wert kann so klein werden, dass er von einem Netz nicht mehr gelernt werden kann. Hier spricht man von einem verschwindenden Gradient.

Das Problem der explodierenden Gradienten lässt sich durch eine sinnvolle Obergrenze beheben. Bei den verschwindenden Gradienten sieht eine Lösung wesentlich schwieriger aus.
Sources: http://deeplearning4j.org/lstm.html

2.2.3 Problem der Langzeit-Abhängigkeiten (unfinished)

Wie bereits erwähnt sind RNNs in der Lage Sequenzen zu erkennen und mit Abhängigkeiten zu arbeiten, doch diese Fähigkeit ist leider begrenzt. Besteht nur eine kleine zeitliche Lücke zwischen den von einander abhängigen Daten, ist ein RNN in der Lage diesen Zusammenhang zu erkennen und die richtigen Schlüsse zu ziehen. Wird der zeitliche Abstand zwischen Eingabe der Daten und dem Zeitpunkt an dem sie für ein Ergebnis benötigt werden jedoch sehr groß kann ein RNN diesen Zusammenhang nicht mehr herstellen. Als Beispiel gibt [Olah 2015] in seinem Artikel ein Sprach-Model an, welches das nächste Wort abhängig vom Vorherigen vorhersagt. Ein RNN ist in der Lage im Satz „Die Wolken sind im Himmel.“ das letzte Wort vorauszusagen, da der Abstand von Himmel und Wolken sehr klein ist. Im Text „Ich bin in Frankreich aufgewachsen. … Ich spreche fließend französisch.“ kann der Abstand zum letzten Wort aber sehr groß sein und die vorherigen Wörter lassen lediglich den Schluss zu das eine Sprache folgen muss. Denn Kontext, dass es sich sehr wahrscheinlich um französisch handelt,

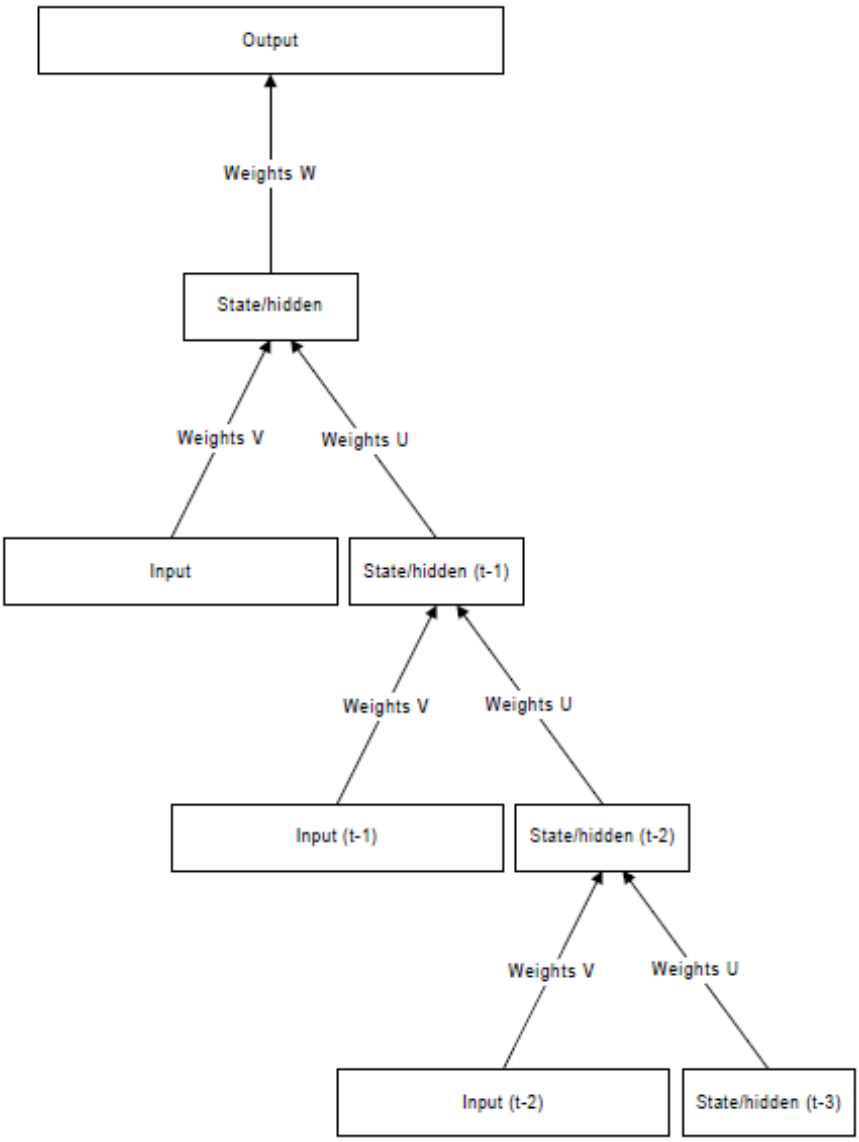


Abb. 2.4: entrolltes RNN für BPTT (Quelle: [1])

erhällt man nur durch den ersten Satz. Ein RNN kann sich aber keinen ganzen Text merken und somit hier den Zusammenhang von Frankreich und französisch nicht lernen.

Um das Problem der Langzeit-Abhängigkeiten zu lösen, benutzt man LSTMs.
Sources: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

2.3 Long Short-Term Memory (LSTM) (unfinished)

LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is practically their default behavior, not something they struggle to learn!

Long Short-Term Memory (LSTM) Netze sind eine besondere Art von Recurrent Netzwerken, die mit Langzeit-Abhängigkeiten arbeiten können. Sie bestehen aus Speicherzellen, in die Informationen geschrieben und wieder herausgelesen werden können. Mit Hilfe von Toren (Gates), die geöffnet oder geschlossen werden, entscheidet eine Zelle was gespeichert wird und wann ein Auslesen, Reinschreiben und Löschen erlaubt ist. Diese Tore sind analog und durch eine Sigmoid-Funktion implementiert, so dass sich ein Bereich von 0 bis 1 ergibt. (Analog hat den Vorteil gegenüber digital dass es differenzierbar ist und somit für die Backpropagation geeignet.) Genau wie die Eingänge bei den Feedforward und Recurrent Netzen besitzen die Tore Gewichte. Diese Gewichte werden ebenfalls während des Lernprozesses angepasst, so dass die Zelle lernt wann Daten eingelassen, ausgelesen oder gelöscht werden.

Abbildung 2.5 zeigt zum Vergleich oben ein simples Recurrent Netz und unten ein LSTM Netz. Beide sind über drei Zeitschritte dargestellt. ... All recurrent neural networks have the form of a chain of repeating modules of neural network. In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer. LSTMs also have this chain like structure, but the repeating module has a different structure. Instead of having a single neural network layer, there are four, interacting in a very special way. In the above diagram, each line carries an entire vector, from the output of one node to the inputs of others. The pink circles represent pointwise operations, like vector addition, while the yellow boxes are learned neural network layers.
Sources: https://colah.github.io/posts/2015-08-Understanding-LSTMs/, http://deeplearning4j.org/lstm.html

LSTMs help preserve the error that can be backpropagated through time and layers. By maintaining a more constant error, they allow recurrent nets to continue to learn over many time steps (over 1000), thereby opening a channel to link causes and effects remotely.

You may wonder why LSTMs have a forget gate when their purpose is to link distant occurrences to a final output. Well, sometimes itâ€™s good to forget. If youâ€™re analyzing a

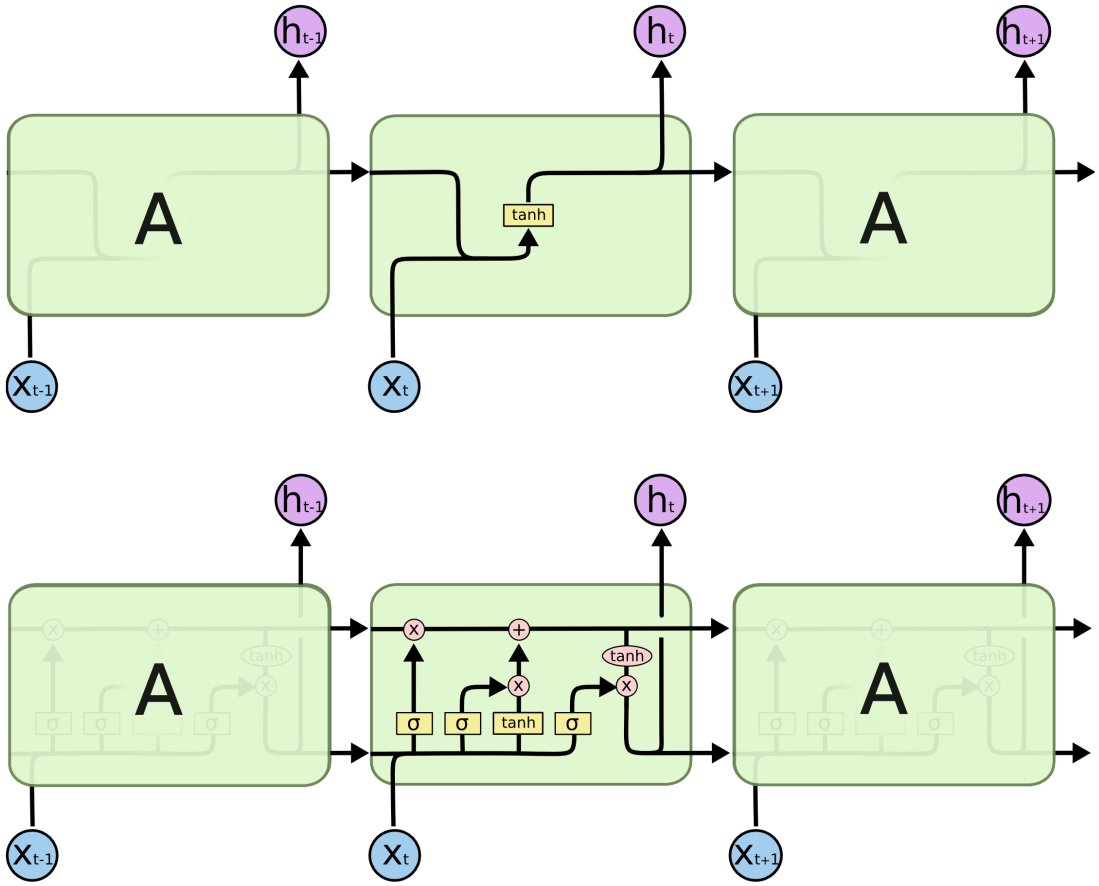


Abb. 2.5: Vergleich RNN und LSTM (Quelle: [2])

text corpus and come to the end of a document, for example, you may have no reason to believe that the next document has any relationship to it whatsoever, and therefore the memory cell should be set to zero before the net ingests the first element of the next document.

Further links: (could be sources for pictures)
http://people.idsia.ch/ juergen/rnn.html
https://karpathy.github.io/2015/05/21/rnn-effectiveness/
https://www.cs.cmu.edu/ bhiksha/courses/deeplearning/Fall.2015/pdfs/Werbos.backprop.pdf
http://www.cs.toronto.edu/ graves/phd.pdf
http://www.felixgers.de/papers/phd.pdf
http://arxiv.org/pdf/1503.04069.pdf

2.3.1 Bauteile einer Speicherzelle

Zellzustand

The key to LSTMs is the cell state, the horizontal line running through the top of the diagram. The cell state is kind of like a conveyor belt. It runs straight down the entire chain, with only some minor linear interactions. Itâ€™s very easy for information to just flow along it unchanged. The LSTM does have the ability to remove or add information to the cell state,

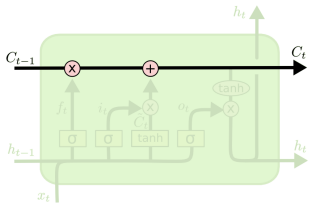
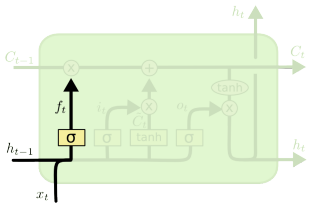


Abb. 2.6: LSTM C line (Quelle: [2])

carefully regulated by structures called gates. Gates are a way to optionally let information through. They are composed out of a sigmoid neural net layer and a pointwise multiplication operation. The sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through. A value of zero means ”let nothing through,”while a value of one means ”let everything through!Än LSTM has three of these gates, to protect and control the cell state.

Forget Gate

The first step in our LSTM is to decide what information weâ€™re going to throw away from the cell state. This decision is made by a sigmoid layer called the ”forget gate layer.Ït looks at htâˆ’1 and xt, and outputs a number between 0 and 1 for each number in the cell state Ctâˆ’1. A 1 represents ”completely keep this”while a 0 represents ”completely get rid of this.”

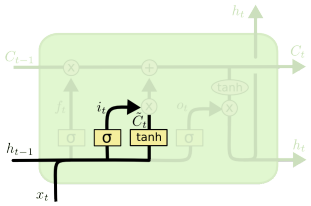


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Abb. 2.7: LSTM focus f (Quelle: [2])

Input gate

The next step is to decide what new information weâ€™re going to store in the cell state. This has two parts. First, a sigmoid layer called the ïnput gate layer”decides which values weâ€™ll update. Next, a tanh layer creates a vector of new candidate values, C t, that could be added to the state. In the next step, weâ€™ll combine these two to create an update to the state.

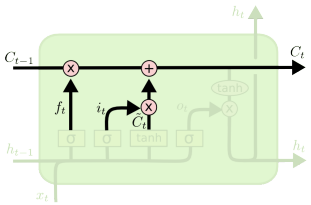


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Abb. 2.8: LSTM focus i (Quelle: [2])

Update Cell state

Itâ€™s now time to update the old cell state, Ctâˆ’1, into the new cell state Ct. The previous steps already decided what to do, we just need to actually do it. We multiply the old state by ft, forgetting the things we decided to forget earlier. Then we add itâˆ—C t. This is the new candidate values, scaled by how much we decided to update each state value.

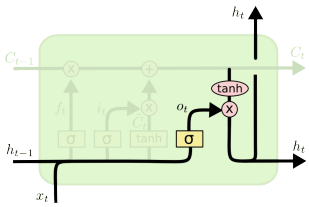


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Abb. 2.9: LSTM focus C (Quelle: [2])

Output

Finally, we need to decide what weâ€™re going to output. This output will be based on our cell state, but will be a filtered version. First, we run a sigmoid layer which decides what parts of the cell state weâ€™re going to output. Then, we put the cell state through tanh (to push the values to be between âˆ’1 and 1) and multiply it by the output of the sigmoid gate, so that we only output the parts we decided to.



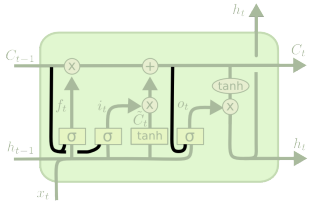
$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Abb. 2.10: LSTM focus o (Quelle: [2])

2.3.2 LSTM Varianten

What Iâ€™ve described so far is a pretty normal LSTM. But not all LSTMs are the same as the above. In fact, it seems like almost every paper involving LSTMs uses a slightly different version. The differences are minor, but itâ€™s worth mentioning some of them.

One popular LSTM variant, introduced by Gers & Schmidhuber (2000), is adding ”peephole connections.”This means that we let the gate layers look at the cell state. The above diagram

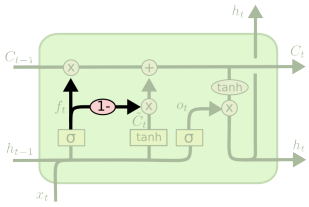


$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t]\ +\ b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t]\ +\ b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t]\ +\ b_o\right)$$

Abb. 2.11: LSTM peepholes (Quelle: [2])

adds peepholes to all the gates, but many papers will give some peepholes and not others.

Another variation is to use coupled forget and input gates. Instead of separately deciding what to forget and what we should add new information to, we make those decisions together. We only forget when weâ€™re going to input something in its place. We only input new values to the state when we forget something older. A slightly more dramatic variation on the LSTM



$$C_t = f_t * C_{t-1} + (1 - \boldsymbol{f_t}) * \tilde{C}_t$$

Abb. 2.12: LSTM tied (Quelle: [2])

is the Gated Recurrent Unit, or GRU, introduced by Cho, et al. (2014). It combines the forget and

input gates into a single üpdate gate.Ït also merges the cell state and hidden state, and makes some other changes. The resulting model is simpler than standard LSTM models, and has been growing increasingly popular. These are only a few of the most notable LSTM variants. There

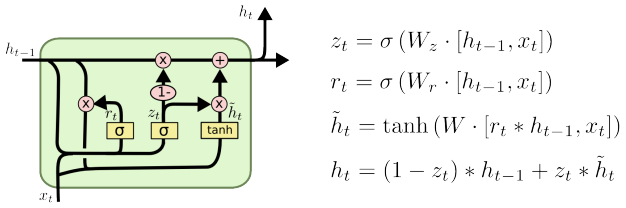


Abb. 2.13: LSTM GRU (Quelle: [2])

are lots of others, like Depth Gated RNNs by Yao, et al. (2015). Thereâ€™s also some completely different approach to tackling long-term dependencies, like Clockwork RNNs by Koutnik, et al. (2014).

Which of these variants is best? Do the differences matter? Greff, et al. (2015) do a nice comparison of popular variants, finding that theyâ€™re all about the same. Jozefowicz, et al. (2015) tested more than ten thousand RNN architectures, finding some that worked better than LSTMs on certain tasks.

2.4 Aktueller Forschungsstand und Problemstellungen (unfinished)

Charakterisierungen ...

http://deeplearning4j.org/neuralnet-overview

As you think about one problem deep learning can solve, ask yourself: What categories do I care about? What information can I act upon? Those outcomes are labels that would be applied to data: spam or not_spam, good_guy or bad_guy, angry_customer or happy_customer. Then ask: Do I have the data to accompany those labels? Can I find labeled data, or can I create a labeled dataset (with a service like Mechanical Turk or Crowdflower) that I can use to teach an algorithm the correlation between labels and inputs?

http://deeplearning4j.org/lstm.html Recurrent nets are a type of artificial neural network designed to recognize patterns in sequences of data, such as text, genomes, handwriting, the spoken word, or numerical times series data emanating from sensors, stock markets and government agencies.

https://colah.github.io/posts/2015-08-Understanding-LSTMs/ In the last few years, there have been incredible success applying RNNs to a variety of problems: speech recognition, language modeling, translation, image captioningâ€¦ The list goes on. Iâ€™ll leave discussion of the amazing feats one can achieve with RNNs to Andrej Karpathyâ€™s excellent blog post, The Unreasonable Effectiveness of Recurrent Neural Networks. Essential to these successes is the use of ”LSTMs,ä very special kind of recurrent neural network which works, for many tasks, much much better than the standard version. Almost all exciting results based on recurrent neural networks are achieved with them.

https://colah.github.io/posts/2015-08-Understanding-LSTMs/ LSTMs were a big step in what we can accomplish with RNNs. Itâ€™s natural to wonder: is there another big step? A common opinion among researchers is: ”Yes! There is a next step and itâ€™s attention!”The idea is to let every step of an RNN pick information to look at from some larger collection of information. For example, if you are using an RNN to create a caption describing an image, it might pick a part of the image to look at for every word it outputs. In fact, Xu, et al. (2015) do exactly this â€“ it might be a fun starting point if you want to explore attention! Thereâ€™s been a number of really exciting results using attention, and it seems like a lot more are around the cornerâ€¦ Attention isnâ€™t the only exciting thread in RNN research. For example, Grid LSTMs by Kalchbrenner, et al. (2015) seem extremely promising. Work using RNNs in generative models â€“ such as Gregor, et al. (2015), Chung, et al. (2015), or Bayer & Osendorfer (2015) â€“ also seems very interesting. The last few years have been an exciting time for recurrent neural networks, and the coming ones promise to only be more so!

http://deeplearning4j.org/lstm.html In the mid-90s, a variation of recurrent net with so-called Long Short-Term Memory units, or LSTMs, was proposed by the German researchers Sepp Hochreiter and Juergen Schmidhuber as a solution to the vanishing gradient problem.

http://deeplearning4j.org/lstm.html In the mid-90s, a variation of recurrent net with so-called Long Short-Term Memory units, or LSTMs, was proposed by the German researchers Sepp Hochreiter and Juergen Schmidhuber as a solution to the vanishing gradient problem.

Quellenverzeichnis

Literatur

[Boden 2001] Mikael Boden. "BPTT: entrolltes RNN". In: (2001). URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.16.6652&rep=rep1&type=pdf (besucht am 16. 07. 2016).

[DL4J] *Deeplearning4j: Open-source distributed deep learning for the JVM*. Version Apache Software Foundation License 2.0. Deeplearning4j Development Team DL4J. URL: http://deeplearning4j.org/ (besucht am 30. 06. 2016).

[Olah 2015] Christopher Olah. *Understanding LSTM Networks*. 2015. URL: https://colah.github.io/posts/2015-08-Understanding-LSTMs/ (besucht am 12. 07. 2016).

Bilder

[1] Mikael Boden. "BPTT: entrolltes RNN". In: (2001). URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.16.6652&rep=rep1&type=pdf (besucht am 16. 07. 2016).

[2] Christopher Olah. URL: https://colah.github.io/posts/2015-08-Understanding-LSTMs/ (besucht am 12. 07. 2016).

[4] Deeplearning4j Development Team. *Feedforward Netz*. URL: http://deeplearning4j.org/neuralnet-overview (besucht am 30. 06. 2016).

[5] Deeplearning4j Development Team. *Feedforward Neuron*. URL: http://deeplearning4j.org/neuralnet-overview (besucht am 30. 06. 2016).

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 1. Januar 2345 Marina Knabbe