WILDML
AI, DEEP LEARNING, NLP

RECURRENT NEURAL NETWORKS TUTORIAL, PART 1 – INTRODUCTION TO RNNS

September 17, 2015

Recurrent Neural Networks (RNNs) are popular models that have shown great promise in many NLP tasks. But despite their recent popularity I've only found a limited number of resources that throughly explain how RNNs work, and how to implement them. That's what this tutorial is about. It's a multi-part series in which I'm planning to cover the following:

1. Introduction to RNNs (this post)
2. Implementing a RNN using Python and Theano
3. Understanding the Backpropagation Through Time (BPTT) algorithm and the vanishing gradient problem
4. Implementing a GRU/LSTM RNN

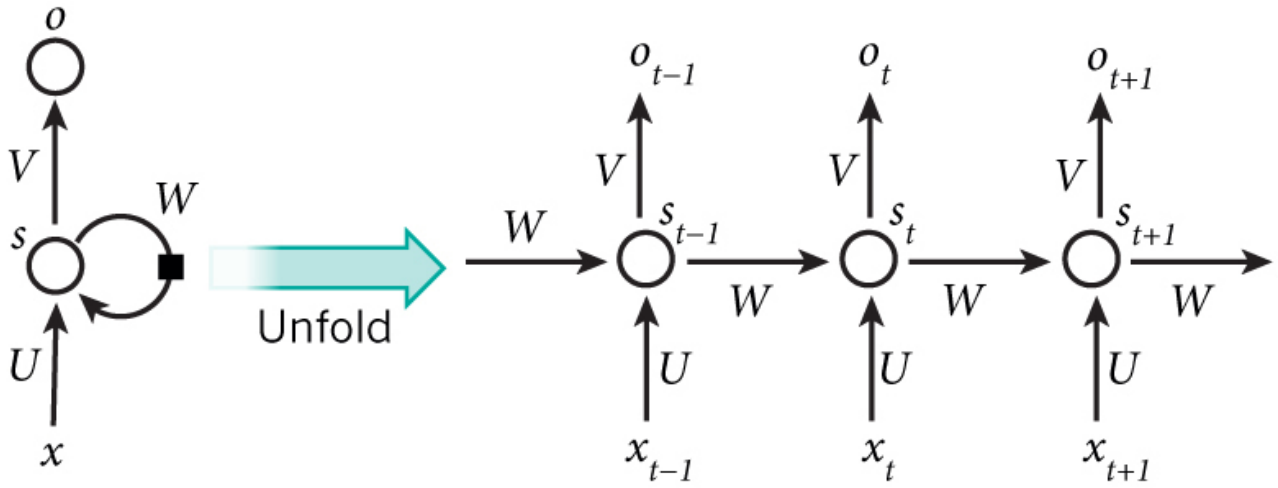
As part of the tutorial we will implement a recurrent neural network based language model. The applications of language models are two-fold: First, it allows us to score arbitrary sentences based on how likely they are to occur in the real world. This gives us a measure of grammatical and semantic correctness. Such models are typically

used as part of Machine Translation systems. Secondly, a language model allows us to generate new text (I think that's the much cooler application). Training a language model on Shakespeare allows us to generate Shakespeare-like text. This fun post by Andrej Karpathy demonstrates what character-level language models based on RNNs are capable of.

I'm assuming that you are somewhat familiar with basic Neural Networks. If you're not, you may want to head over to Implementing A Neural Network From Scratch, which guides you through the ideas and implementation behind non-recurrent networks.

WHAT ARE RNNS?

The idea behind RNNs is to make use of sequential information. In a traditional neural network we assume that all inputs (and outputs) are independent of each other. But for many tasks that's a very bad idea. If you want to predict the next word in a sentence you better know which words came before it. RNNs are called *recurrent* because they perform the same task for every element of a sequence, with the output being depended on the previous computations. Another way to think about RNNs is that they have a "memory" which captures information about what has been calculated so far. In theory RNNs can make use of information in arbitrarily long sequences, but in practice they are limited to looking back only a few steps (more on this later). Here is what a typical RNN looks like:



A recurrent neural network and the unfolding in time of the computation involved in its forward computation. Source: Nature

The above diagram shows a RNN being *unrolled* (or unfolded) into a full network. By unrolling we simply mean that we write out the network for the complete sequence. For example, if the sequence we care about is a sentence of 5 words, the network would be unrolled into a 5-layer neural network, one layer for each word. The formulas that govern the computation happening in a RNN are as follows:

- x_t is the input at time step t. For example, x_1 could be a one-hot vector corresponding to the second word of a sentence.
- s_t is the hidden state at time step t. It's the "memory" of the network. s_t is calculated based on the previous hidden state and the input at the current step: $s_t = f(Ux_t + Ws_{t-1})$. The function f usually is a nonlinearity such as tanh or ReLU. s_{-1}, which is required to calculate the first hidden state, is typically initialized to all zeroes.
- o_t is the output at step t. For example, if we wanted to predict the next word in a sentence it would be a vector of probabilities across our vocabulary. $o_t = \text{softmax}(Vs_t)$.

There are a few things to note here:

- You can think of the hidden state s_t as the memory of the network. s_t captures

information about what happened in all the previous time steps. The output at step o_t is calculated solely based on the memory at time t. As briefly mentioned above, it's a bit more complicated in practice because s_t typically can't capture information from too many time steps ago.

- Unlike a traditional deep neural network, which uses different parameters at each layer, a RNN shares the same parameters (U, V, W above) across all steps. This reflects the fact that we are performing the same task at each step, just with different inputs. This greatly reduces the total number of parameters we need to learn.
- The above diagram has outputs at each time step, but depending on the task this may not be necessary. For example, when predicting the sentiment of a sentence we may only care about the final output, not the sentiment after each word. Similarly, we may not need inputs at each time step. The main feature of an RNN is its hidden state, which captures some information about a sequence.

WHAT CAN RNNS DO?

RNNs have shown great success in many NLP tasks. At this point I should mention that the most commonly used type of RNNs are LSTMs, which are much better at capturing long-term dependencies than vanilla RNNs are. But don't worry, LSTMs are essentially the same thing as the RNN we will develop in this tutorial, they just have a different way of computing the hidden state. We'll cover LSTMs in more detail in a later post. Here are some example applications of RNNs in NLP (by non means an exhaustive list).

LANGUAGE MODELING AND GENERATING TEXT

Given a sequence of words we want to predict the probability of each word given the previous words. Language Models allow us to measure how likely a sentence is, which is an important input for Machine Translation (since high-probability sentences are typically correct). A side-effect of being able to predict the next word is that we get a

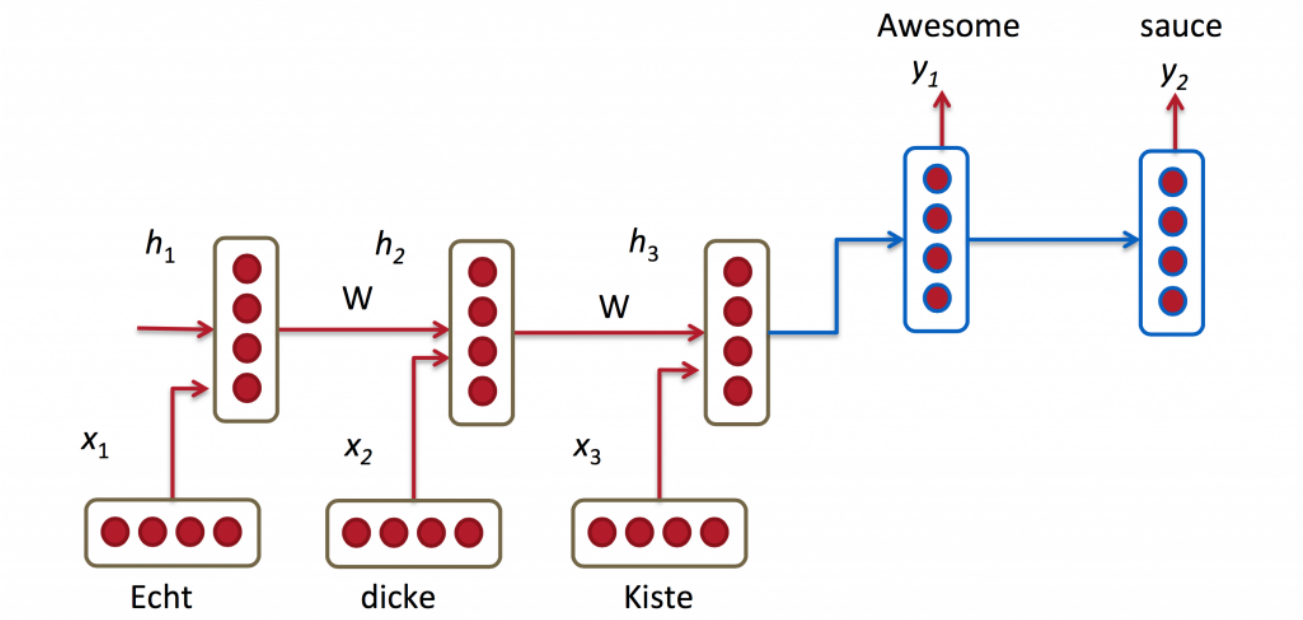
generative model, which allows us to generate new text by sampling from the output probabilities. And depending on what our training data is we can generate all kinds of stuff. In Language Modeling our input is typically a sequence of words (encoded as one-hot vectors for example), and our output is the sequence of predicted words. When training the network we set $o_t = x_{t+1}$ since we want the output at step t to be the actual next word.

Research papers about Language Modeling and Generating Text:

- Recurrent neural network based language model
- Extensions of Recurrent neural network based language model
- Generating Text with Recurrent Neural Networks

MACHINE TRANSLATION

Machine Translation is similar to language modeling in that our input is a sequence of words in our source language (e.g. German). We want to output a sequence of words in our target language (e.g. English). A key difference is that our output only starts after we have seen the complete input, because the first word of our translated sentences may require information captured from the complete input sequence.



RNN for Machine Translation. Image Source: http://cs224d.stanford.edu/lectures/CS224d-Lecture8.pdf

Research papers about Machine Translation:

- A Recursive Recurrent Neural Network for Statistical Machine Translation
- Sequence to Sequence Learning with Neural Networks
- Joint Language and Translation Modeling with Recurrent Neural Networks

SPEECH RECOGNITION

Given an input sequence of acoustic signals from a sound wave, we can predict a sequence of phonetic segments together with their probabilities.

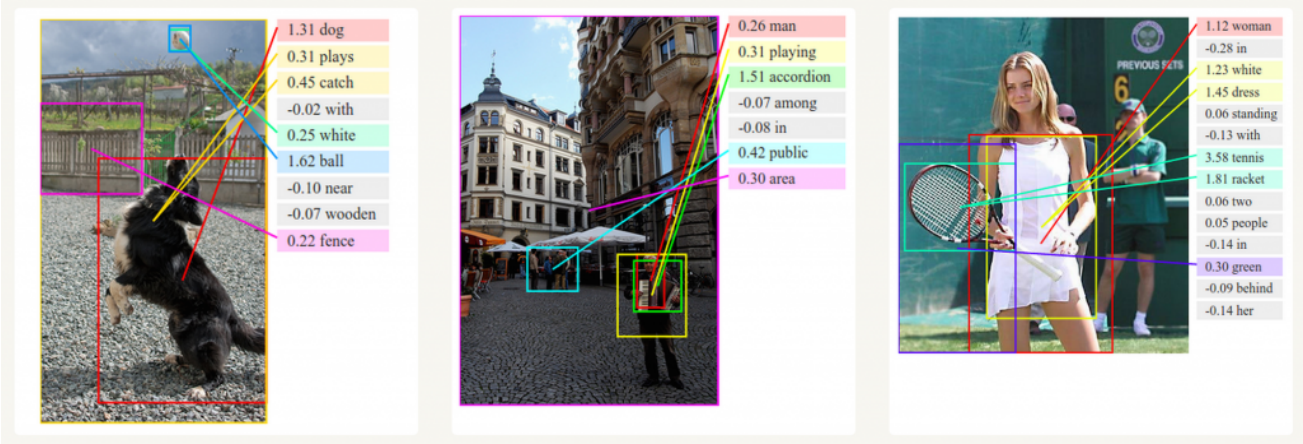
Research papers about Speech Recognition:

- Towards End-to-End Speech Recognition with Recurrent Neural Networks

GENERATING IMAGE DESCRIPTIONS

Together with convolutional Neural Networks, RNNs have been used as part of a

model to generate descriptions for unlabeled images. It's quite amazing how well this seems to work. The combined model even aligns the generated words with features found in the images.



Deep Visual-Semantic Alignments for Generating Image Descriptions. Source: http://cs.stanford.edu/people/karpathy/deepimagesent/

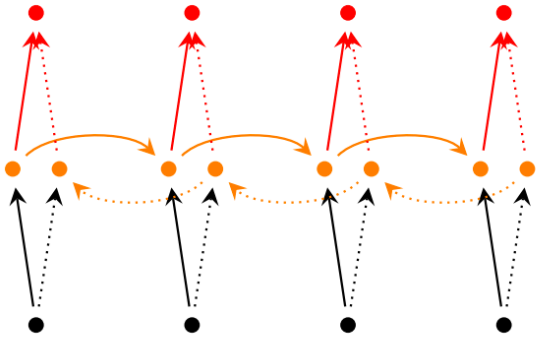
TRAINING RNNS

Training a RNN is similar to training a traditional Neural Network. We also use the backpropagation algorithm, but with a little twist. Because 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. For example, in order to calculate the gradient at $t = 4$ we would need to backpropagate 3 steps and sum up the gradients. This is called Backpropagation Through Time (BPTT). If this doesn't make a whole lot of sense yet, don't worry, we'll have a whole post on the gory details. For now, just be aware of the fact that vanilla RNNs trained with BPTT have difficulties learning long-term dependencies (e.g. dependencies between steps that are far apart) due to what is called the vanishing/exploding gradient problem. There exists some machinery to deal with these problems, and certain types of RNNs (like LSTMs) were specifically designed to get around them.

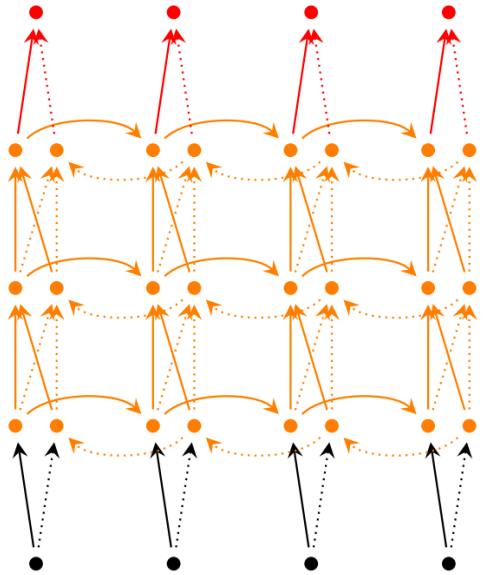
RNN EXTENSIONS

Over the years researchers have developed more sophisticated types of RNNs to deal with some of the shortcomings of the vanilla RNN model. We will cover them in more detail in a later post, but I want this section to serve as a brief overview so that you are familiar with the taxonomy of models.

Bidirectional RNNs are based on the idea that the output at time t may not only depend on the previous elements in the sequence, but also future elements. For example, to predict a missing word in a sequence you want to look at both the left and the right context. Bidirectional RNNs are quite simple. They are just two RNNs stacked on top of each other. The output is then computed based on the hidden state of both RNNs.



Deep (Bidirectional) RNNs are similar to Bidirectional RNNs, only that we now have multiple layers per time step. In practice this gives us a higher learning capacity (but we also need a lot of training data).



LSTM networks are quite popular these days and we briefly talked about them above. LSTMs don't have a fundamentally different architecture from RNNs, but they use a different function to compute the hidden state. The memory in LSTMs are called *cells* and you can think of them as black boxes that take as input the previous state h_{t-1} and current input x_t. Internally these cells decide what to keep in (and what to erase from) memory. They then combine the previous state, the current memory, and the input. It turns out that these types of units are very efficient at capturing long-term dependencies. LSTMs can be quite confusing in the beginning but if you're interested in learning more this post has an excellent explanation.

CONCLUSION

So far so good. I hope you've gotten a basic understanding of what RNNs are and what they can do. In the next post we'll implement a first version of our language model RNN using Python and Theano. Please leave questions in the comments!

Posted in: Deep Learning, Neural Networks, Recurrent Neural Networks

← *Speeding up your Neural Network with Theano and the GPU*

Recurrent Neural Networks Tutorial, Part 2 – Implementing a RNN with Python, Numpy and Theano →

Michael David Watson • a year ago

So what exactly are U, V and W? You mention that they are the same parameters for each step, and that state and output are dependent on them, but I am not sure if they are just place holders for example paramaters or have a specific meaning.

5 • Reply • Share ›

Denny Britz Mod → Michael David Watson • a year ago

U, V and W are the parameters of the network that you need to learn from your training data. So before you've trained your network using data you don't usually know the "content" of these matrices.

Here's an example for Language Modeling for when you have 5 words in your vocabulary: I, am, the, super, man.

- Your input vectors x would be of dimension 5, essentially selecting a word, e.g. [0 0 1 0 0] ("the")

- Together U and W define how to calculate the new memory of the network given the previous memory and the input word (together with the function f). You start out with randomly initialized U and W matrices, but as you train your network U will learn how to map the word vector above into the "space" of the hidden layer, so it would be of size [D]x5, and after multiplication with x you get a result of size [D]. You can pick [D], it's how big you want the hidden state state to be.

- V defines how to map the hidden state (memory) back into the space of possible words, so it's of size 5x[D]. Multiplying V and s would give you a

vector of scores for the predicted word, e.g. [2313, 31, 11, 55, 113] and the softmax would convert these scores into probabilities. The word with the highest probability (the first element in the above example) would then become your prediction.

Does that make sense?

3 ^ v • Reply • Share ›

skotadi ➚ Denny Britz • a year ago

So U is the "Word Vector to Hidden Space map", and V is the "Hidden Space to Word Vector map." What would a good name for W be in this context?

1 ^ v • Reply • Share ›

Denny Britz Mod ➚ skotadi • a year ago

You can think of it as determining which parts of the previous memory (hidden state) to use, and how to use them. For example, considering that previous word was a verb. But W*s and U*x are added together, so they interact and their "meaning" can't be considered in isolation. It's the combination of the two that makes up the hidden state / memory.

3 ^ v • Reply • Share ›

Xiaokai Zhao ➚ Denny Britz • a year ago

Thanks for your explanation, I have a question why it is W*s+U*x instead of some other operators? Why taking sum? Thank you!

1 ^ v • Reply • Share ›

Denny Britz Mod ➚ Xiaokai Zhao • a year ago

I don't think there's a principled reason for why it's the sum. It's simple and fast to compute :) But it could also be something more complex. For example, in Neural Tensor Networks (http://nlp.stanford.edu/~soche... you're multiplying with a tensor instead of just doing the sum to model more complex interactions than a sum could.

1 ^ v • Reply • Share ›

skotadi ➚ Denny Britz • a year ago

Sure, that's understood. I know they can't be used in isolation; I'm interested in giving the properties better names for improved code understanding. So W as something like "Previous Hidden State Adjudicator" seems like it could fit. That sound right?

Denny Britz Mod ↗ skotadi • a year ago

Sounds right :)

Kyuhyoung Choi ↗ Denny Britz • a year ago

Thanks a lot for this great blog. Reading your reply, I got two questions.
1. So the input of the function f (tanh or Relu) is a vector of length [D]. Should'nt the input of f be a scalar?
2. In your example, [2313, 31, 11, 55, 113] gives the highest probability to the word "I". Right? In this sense, does the input vector x (such as [0 1 0 0 1]) have probabilistic meaning so that I can imagine such an input x as [0 0.8 0 0.2 1]? I thought the input x is just an indicator vector in which the magnitude of the nonzero element makes no difference.

Denny Britz Mod ↗ Kyuhyoung Choi • a year ago

Hi Kyuhyoung,

1. Yep, the input of f is a vector. This just means that we apply f to each element independetly, and the output is a vector as well. Each hidden state s_t is a vector, or layer of neurons. Each element of the [D] vector is the activation value of a neuron in that layer.

2. Yes, I believe you can put a probabilistic meaning to the input vector X, though I've never tried that and intuitively I'm not sure what that would correspond in terms of a sentence (maybe interchangable words?). It is not simply an indicator vector because we are multiplying it with U, and U is a matrix that we are learning from the data. So by using a different x you would learn a different mapping U.

The same is true for the "correct" outputs (the training labels y) that you are using to train the network. We set these to one-hot vectors because we exactly know which word they should be (so the probability is 100%), but they also have a probabilistic interpretation. I think that'll become clearer when you look at the code in the next part and see how we calculate the error.

One thing to consider is that one-hot vectors have a sparse representation, so they're very memory efficient.

Reply • Share ›

Nick Byrne • a year ago

Thanks for a great intro to RNNs, I look forward to the following ones.
Have you seen any good 'side by side' comparisons of *NN algorithms performance?

2 Reply • Share ›

Denny Britz Mod ➚ Nick Byrne • a year ago

Thanks Nick! It's bit difficult to compare *NN architectures because many of them solve fundamentally different problems. For example, you can't use traditional Deep Neural Networks to solve the problems that RNNs solve. I haven't seen any papers that do a detailed comparison, but there probably are some, I just don't know about them. A quick Google search turned up this (relatively new) one: http://jmlr.org/proceedings/pa... - It compares several architectures (mostly LSTMs) across wide range of hyperparameters.

Edit: Also, I just remembered this one (again, mostly concerned with LSTMs): http://arxiv.org/pdf/1503.0406...

1 Reply • Share ›

Zozozoz • a year ago

Great post for me, can't wait for the next one.

2 Reply • Share ›

Jubei • 10 months ago

First of all thanks for the great tutorial. I have a couple of questions:

"x_t is the input at time step t. For example, x_1 could be a one-hot vector corresponding to the second word of a sentence." So the first word of a sentence would be x_0?

would be x_0?

"When training the network we set o_t = x_{t+1} since we want the output at step t to be the actual next word." I don't understand this statement. Why equate an output directly with the next input, ignoring all the hidden layers? Shouldn't the output o_t be dependent on the hidden layers even when training the network?

In the section where you explain a taxonomy of models you have a figure. Could you please clarify what each colour is? Is it Black: input, Orange: Hidden Layer, Red: Output?

Thanks in advance.

see more

1 • Reply • Share ›

Denny Britz Mod → Jubei • 10 months ago

Hi Jubei,

> So the first word of a sentence would be x_0

Yes, that's right.

> I don't understand this statement. Why equate an output directly with the next input, ignoring all the hidden layers? Shouldn't the output o_t be dependent on the hidden layers even when training the network?

Yes, you're right. The output *is* dependent on the hidden layers, but you need a supervision signal that tells the network what would be the correct output would be. The hidden layers compute the output, but you're using the actual next word to train the network. Does that make sense?

> In the section where you explain a taxonomy of models you have a figure. Could you please clarify what each colour is? Is it Black: input, Orange: Hidden Layer, Red: Output?

Yes, that's right. Black: input, Orange: Hidden Layer, Red: Output.

• Reply • Share ›

Jubei → Denny Britz • 10 months ago

>you're using the actual next word to train the network

Yes it makes sense now. Thank you.

Reply • Share ›

Martyn Mlostekk • a year ago

Great introduction, cant wait for the next part!

1 Reply • Share ›

Denny Britz Mod → Martyn Mlostekk • a year ago

Thanks Martyn, glad to hear that!

Reply • Share ›

Mark Wissler • a year ago

This is a great post. Very exciting for the next one.

1 Reply • Share ›

Denny Britz Mod → Mark Wissler • a year ago

Thanks Mark! Glad you liked it!

Reply • Share ›

William Davis • a year ago

Wow. Great article. Cant wait for the next post. This is an area of CS I've not had much exposure, but after reading this I'm geeking out a little.

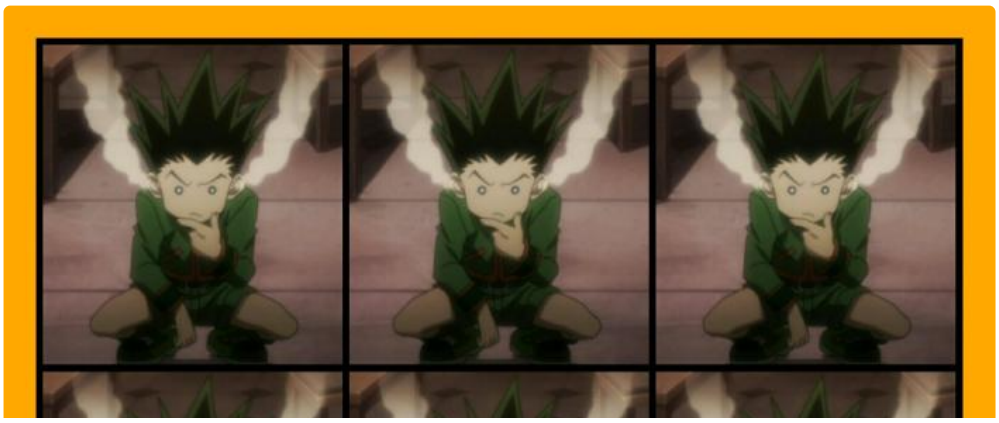
1 Reply • Share ›

Denny Britz Mod → William Davis • a year ago

Glad to hear that! And thanks for reading :)

Reply • Share ›

Achraf Benlemkaddem • 4 days ago



• Reply • Share ›

Sandrine • 14 days ago

Thx for introduction to the RNN, am asking how can i use it to generate Shakespeare's writing

• Reply • Share ›

Haisen Guo • 17 days ago

Nice explanation!!! But what is the boundary between long and short? Do states ten cycles away belong to short or long memory?

• Reply • Share ›

Ravi Teja • a month ago

Can a recurrent neural network be used to learn a sequence with slightly different variations? For example, could I get an RNN trained so that it could produce a sequence of consecutive integers or alternate integers if I have enough training data?

For example, if I train using
1,2,3,4
2,3,4,5
3,4,5,6 and so on
and also train the same network using
1,3,5,7
2,4,6,8
3,5,7,9 and so on,
would I be able to predict both sequences successfully for the test set?

What if I have even more variations in the training data like sequences of every three integers or every four integers, et cetera?

Reply • Share ›

Jeff • a month ago

Hi and thanks for the article! I have a question: in your picture showing the unfolded network there are only connections between t-1 and t, but no direct connection between between t-1 and t+1. Does that mean that t+1 has no information about t-1 and can only look back in time for one step?

Reply • Share ›

Denny Britz Mod → Jeff • a month ago

Yes, it only depends on the state of the previous time step. However, the state at time t contains information about t-1, so it can model dependencies that go back in time for many steps.

Reply • Share ›

leila kerkeni • 2 months ago

First thank for your great tutorial, it's great help for me.
I have x speech signals, I want to classify them according to 7 classes. I have some questions:

- the number of output node must be equal of the number of classes ?
- What's the number of input node ? an input of each speech signal ? Or a matrix for all vectors?
(knowing that each signal is a vector)
- and what's the number of hidden layer?

Thanks in advance.

Reply • Share ›

Akshay • 5 months ago

Thanks for the article, loving the series. Is there a concept of adding more hidden layers (memory units) in RNN? Comparing it to CNN - where as we go deep in network, features learned by hidden layers get more advanced. Would adding more memory units in RNNs be beneficial?

Reply • Share ›

Denny Britz Mod → Akshay • 5 months ago

Yes, you can add more layers and/or make the hidden layers larger. In practice, people use anywhere from 1-8 layers depending on how complex the problem is and how much data is available.

1 • Reply • Share ›

Nhan Vu • 6 months ago

Awesome post. One thing, this may be a typo => "In turns out that these types of units are very efficient at capturing long-term dependencies". Should be "It turns out ...", right? May be this post is generated from a RNN trained from your text, and that is one of its errors, :D.

• Reply • Share ›

Denny Britz Mod ➦ Nhan Vu • 6 months ago

Yep, fixed. Thanks!

• Reply • Share ›

Ashis Samal • 6 months ago

Excellent Tutorial . Thank you very much .
I have a doubt here in this line . "You can pick [D], it's how big you want the hidden state state to be."
Could you please explain 'D' and size of hidden state wrt D ?

• Reply • Share ›

Neel Shah • 6 months ago

Hi Denny, the tutorial on RNN is really amazing. I am still not able to understand the BPTT part that properly. Can you recommend any paper or book or any source that can help me clear my doubts on how bptt actually works. Thanks.

• Reply • Share ›

Rolan Veron Cruz • 6 months ago

Thank you so much for putting in the time and effort to put this tutorial series together! Can't wait for more!

• Reply • Share ›

audakel • 7 months ago

Great post, thanks for keeping the length down.

• Reply • Share ›

mina khoshdeli • 7 months ago

Thank you, it is great!

Reply • Share ›

Vignesh Ungrapalli • 7 months ago

Very well written article. Thanks!!

Reply • Share ›

Irene Li • 7 months ago

That is an impressive post! I saw there are implementations on Python lol Do you have a tensorflow version on it plz!!!!

Reply • Share ›

Denny Britz Mod ↗ Irene Li • 7 months ago

Unfortunately not, but I've been thinking of rewriting the tutorial in Tensorflow and incorporating all the feedback... would that be useful?

Reply • Share ›

Pranjal Daga ↗ Denny Britz • 5 months ago

Definitely, that'd be awesome **@Denny Britz**!

Reply • Share ›

Fatemeh • 9 months ago

Hi there
I have a question.
Is it possible to use the code for time series forecasting and how I can do it?
Do you anyone who is working on this application of RNN?

Reply • Share ›

Denny Britz Mod ↗ Fatemeh • 9 months ago

Yes, RNNs model time series. All applications of RNNs are time series ;)

Reply • Share ›

Xinrui Zhang • 9 months ago

First thank for your great tutorial, it's great help for me.After reading your blog,I have some questions.
Do you think the RNN(or variants of RNN) can be used in text correction(I think predict a missing word is a breakthrough, but I don't how to go on next)? And how?
Thank you.

Reply • Share ›

Denny Britz Mod ↗ Xinrui Zhang • 9 months ago

Yes, text corrections seems like a good fit for RNNs. It's difficult to give specific advice without more information. What would be the input and output of the system you imagine?

^ | v • Reply • Share ›

Xinrui Zhang ↗ Denny Britz • 9 months ago

input:an incorrect sentence(maybe missing a word or a superfluous word or a grammatically incorrect sentence,etc.)
output:a complete and grammatically correct sentence

^ | v • Reply • Share ›

Marwen Wn • 10 months ago

Thanks for this amazing tutorial. I have a question concerning the words vectors in case of a Sentiment Analysis task: Are both these vectors and the weights of the RNN outputs of the training here? In this case, when "using" the RNN for classifing a new text, which is a bunch of words, we can use the representations from the output for words that participated in the training, what about the new words (those who did not appear in the training data set), how do we represent them?

^ | v • Reply • Share ›

Denny Britz Mod ↗ Marwen Wn • 10 months ago

Great question, dealing with out of vocabulary (OOV) words is a common problem. The easiest approach is to simply replace all words that don't occur often enough in the training set with a special "OOV" token, and treat them all the same. It's a pretty reasonable assumption that words that never occur during training are also pretty rare in the new text and so it won't hurt performance too much.

The more sophisticted solution is to use characters as input and then "learn" how words are made up of characters. That allows the network to generalize to unseen words that have common roots with other words, etc. For example: http://arxiv.org/abs/1602.0036...

^ | v • Reply • Share ›

Marwen Wn ↗ Denny Britz • 10 months ago

Do you mean that we can represent these words with the representation of one of the rare words in the training corpus ?

^ | v • Reply • Share ›

Load more comments

ALSO ON **WILDML**

Implementing a Neural Network from Scratch in Python – An Introduction

101 comments • a year ago•

electrocured — Hi, great guide, thanks a lot.For the not initiated, i would like to add the next lines. Took me a while to …

Deep Learning Glossary

10 comments • a year ago•

Federico Magliani — Maybe you can add at the glossary "Data augmentation" and the related techniques for its …

Deep Learning for Chatbots, Part 1 – Introduction

39 comments • 9 months ago•

Emilio Srougo — Thanks for the info! Yes, it seems that api.ai and wit are quite similar, their power lies in making the …

Recurrent Neural Networks Tutorial, Part 2 – Implementing a RNN with …

197 comments • a year ago•

Nitin Agarwal — Thanks for such great post. I was facing problem in understanding the working of …

- CONNECT -

- RECENT POSTS -

Learning Reinforcement Learning (with Code, Exercises and Solutions)

RNNs in Tensorflow, a Practical Guide and Undocumented Features

Deep Learning for Chatbots, Part 2 – Implementing a Retrieval-Based Model in Tensorflow

Deep Learning for Chatbots, Part 1 – Introduction

Attention and Memory in Deep Learning and NLP

Implementing a CNN for Text Classification in TensorFlow

Understanding Convolutional Neural Networks for NLP

Recurrent Neural Network Tutorial, Part 4 – Implementing a GRU/LSTM RNN with Python and Theano

- LINKS -

Home

About

NLP Consulting

- ARCHIVES -

October 2016

August 2016

July 2016

April 2016

January 2016

December 2015

November 2015

October 2015

September 2015

- CATEGORIES -

Conversational Agents

Convolutional Neural Networks

Deep Learning

GPU

Language Modeling

Memory

Neural Networks

NLP

Recurrent Neural Networks

Reinforcement Learning

RNNs

Tensorflow

- SUBSCRIBE TO BLOG VIA EMAIL -

Enter your email address to subscribe to this blog and receive notifications of new posts by email.

Email Address

SUBSCRIBE

- META -

Log in

Entries RSS

Comments RSS

WordPress.org