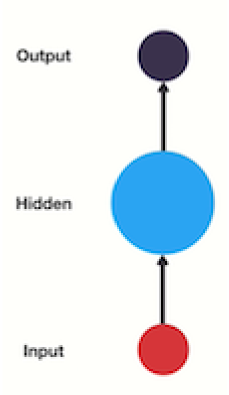
Training and Analyzing Deep Recurrent Neural Networks

Purpose

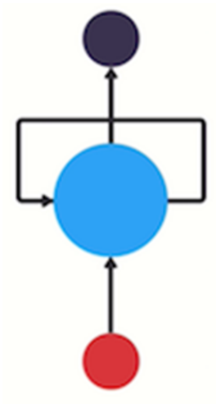
Time series often have a temporal hierarchy, with information that is spread out over multiple time scales. Common recurrent neural networks, do not explicitly accommodate any temporal hierarchy. For RNNs, the primary function of the layers is to introduce memory, not hierarchical processing. In this paper we study the effect of a hierarchy of recurrent neural networks on processing time series. Here, each layer is a recurrent network which receives the hidden state of the previous layer as input. This architecture allows us to perform hierarchical processing on difficult temporal tasks, and more naturally capture the structure of time series. We show that they reach the best performance for recurrent networks in character-level language modelling when trained with simple stochastic gradient descent.

Introduction

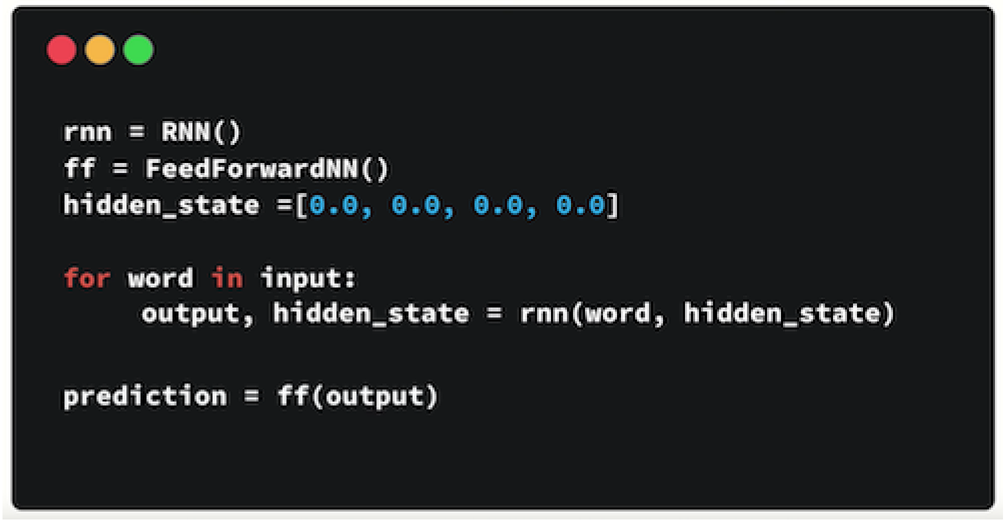
The DNN can be seen as a processing pipeline, in which each layer solves a part of the task before passing it on to the next, until finally the last layer provides the output. The RNN, when folded out in time, it can be considered as a DNN with indefinitely many layers.



Feed forward neural network



Recurrent Neural Network



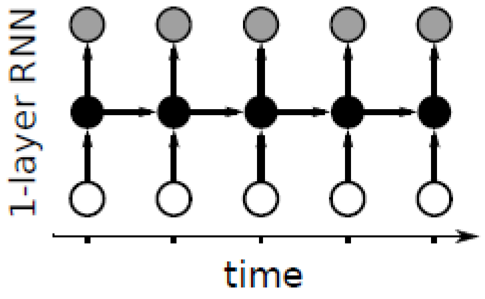
```
rnn = RNN()
ff = FeedForwardNN()
hidden_state =[0.0, 0.0, 0.0, 0.0]

for word in input:
     output, hidden_state = rnn(word, hidden_state)

prediction = ff(output)
```

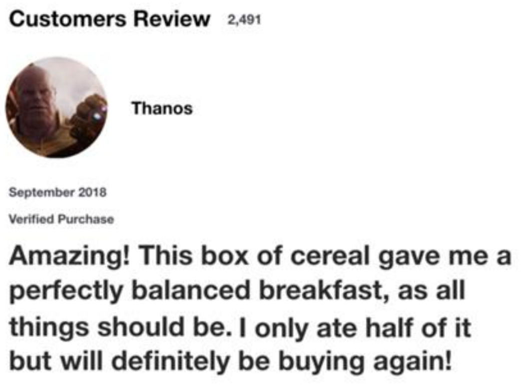
Pseudo code for RNN control flow

For RNNs, the primary function of the layers is to introduce
memory, not hierarchical processing. New information is added in every 'layer' (every network iteration),
and the network can pass this information on for an indefinite number of network updates,
essentially providing the RNN with unlimited memory depth. The network updates do not provide hierarchical
processing of the information except that older data (provided several time
steps ago) passes through the recursion more often. There is no compelling reason why older data
would require more processing steps (network iterations) than newly received data.



The recurrent weights in an RNN learn during the training phase to select what information they need to pass onwards, and what they need to discard.(Long Short-term memory (LSTM) architecture).

Intuition of how LSTM works



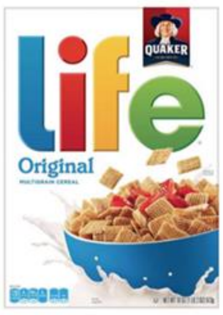
Customers Review 2,491

Thanos

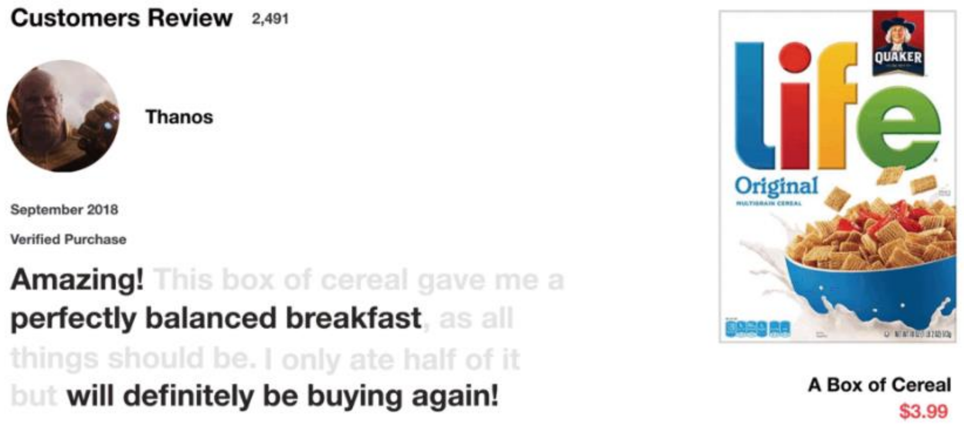
September 2018

Verified Purchase

Amazing! This box of cereal gave me a perfectly balanced breakfast, as all things should be. I only ate half of it but will definitely be buying again!



A Box of Cereal
$3.99


Customers Review 2,491

Thanos

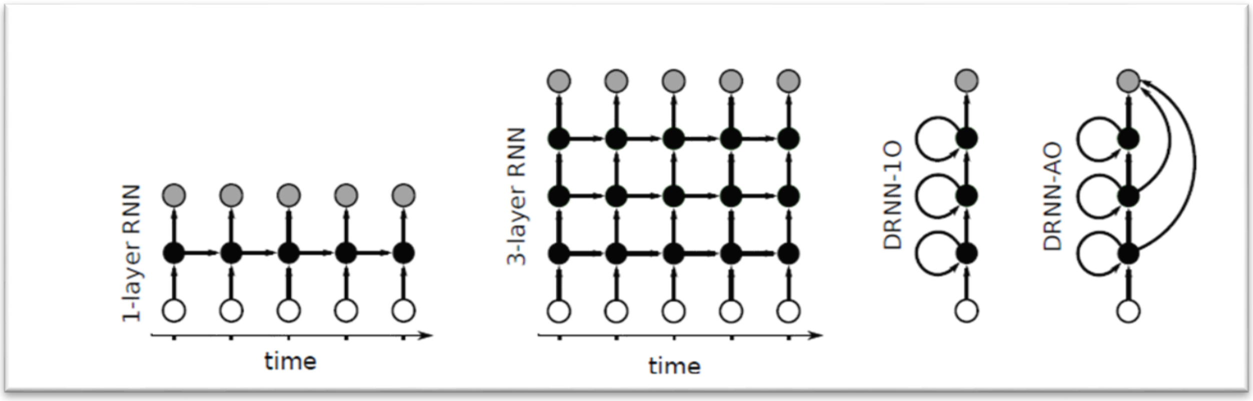
September 2018

Verified Purchase

Amazing! This box of cereal gave me a perfectly balanced breakfast, as all things should be. I only ate half of it but will definitely be buying again!

A Box of Cereal

$3.99



Left:
Standard RNN, folded out in time.
Middle: DRNN of 3 layers folded out in time. Each layer can
be interpreted as an RNN that receives the time series of the previous layer as input.
Right: The two alternative architectures that we study in this paper, where the looped arrows represent the recurrent weights. Either only the top layer connects to the output (DRNN-1O), or all layers do (DRNN-AO).

One potential weakness of a common RNN is that we may need complex, hierarchical processing of the current network input, but this information only passes through one layer of processing before going to the output. Secondly, we may need to process the time series at several time scales. If we consider for example speech, at the lowest level it is built up of phonemes, which exist on a very short time-scale. Next, on increasingly longer time scales, there are syllables, words, phrases, clauses, sentences, and at the highest level for instance a full conversation.

The architecture we study in this paper is essentially a common DNN (a multilayer perceptron) with temporal feedback loops in each layer, which we call a deep recurrent neural network (DRNN). This basically combines the concept of DNNs with RNNs. Each layer
in the hierarchy is a recurrent neural network, and each subsequent layer receives the hidden state of the previous layer as input time series. As we will show, stacking RNNs automatically creates different time scales at different levels, and therefore a temporal hierarchy.

DRNNs embed these different timescales
directly in their structure, and they are able to model long-term dependencies. Using only stochastic gradient descent (SGD) we are able to get state-of-the-art performance for recurrent networks on a Wikipedia-based text corpus.

Deep RNNs ---

1.Hidden state evolution

We define a DRNN with L layers, and N neurons per layer. Suppose we have an input time series s(t) of dimensionality N_{in}, and a target time series y_(t). We denote the hidden state of the i-th layer with a_i(t). W_i and Z_i are the recurrent connections and the connections from the lower layer or input time series, respectively. Its update equation is given by:

$$\begin{aligned}\mathbf{a}_i(t) &= \tanh\left(\mathbf{W}_i\mathbf{a}_i(t-1) + \mathbf{Z}_i\bar{\mathbf{a}}_{i-1}(t)\right) \quad \text{if } i > 1\\ \mathbf{a}_i(t) &= \tanh\left(\mathbf{W}_i\mathbf{a}_i(t-1) + \mathbf{Z}_i\bar{\mathbf{s}}(t)\right) \quad \text{if } i = 1.\end{aligned}$$

2.Generating output

The DRNN generates an output which we denote by y(t). We will consider two scenarios: that where only the highest layer in the hierarchy couples to the output (DRNN-1O), and that where all layers do (DRNN-AO). In the two respective cases, y(t) is given by:

$$\mathbf{y}(t) = \text{softmax}\left(\mathbf{U}\bar{\mathbf{a}}_L(t)\right),$$

where $\mathbf{U}$ is the matrix with the output weights, and

$$\mathbf{y}(t) = \text{softmax}\left(\sum_{i=1}^{L}\mathbf{U}_i\bar{\mathbf{a}}_i(t)\right),$$

such that U_i corresponds to the output weights of the i-th layer.

DRNNs suffer from a pathological curvature in the cost function. If we use backpropagation through time, the error will propagate from the top layer down the hierarchy, but it will have diminished in magnitude once it reaches the lower layers, such that they are not trained effectively. Adding output connections at each layer amends this problem to some degree as the training error reaches all layers directly. Secondly, We can for instance measure the decay of performance by leaving out an individual layer's contribution.

3.Training setup

To avoid extremely large gradients, we applied the often-used trick of normalizing the gradient before using it for weight updates. This simple heuristic seems to be effective to prevent gradient explosions and sudden jumps of the parameters, while not diminishing the end performance. We write the no. of batches we train on as T. The learning rate is set at an initial value η_0, and drops linearly with each subsequent weight update. Suppose $\boldsymbol{\theta}(j)$ is the set of all trainable parameters after j updates, and $\nabla_{\boldsymbol{\theta}}(j)$ is the gradient of a cost function w.r.t. this parameter set, as computed on a randomly sampled part of the training set. Parameter updates are given by:

$$\boldsymbol{\theta}(j+1) = \boldsymbol{\theta}(j) - \eta_0\left(1 - \frac{j}{T}\right)\frac{\nabla_{\boldsymbol{\theta}}(j)}{||\nabla_{\boldsymbol{\theta}}(j)||}. \qquad (3)$$

In the case where we use output connections at the top layer only, we use an incremental layer-wise method to train the network, When adding a layer, the previous output weights are discarded and new output weights are initialised connecting from the new top layer. Over the course of each of these training stages we train the full network with BPTT and linearly reducing the learning rate to zero before a new layer is added. We always train the full network after each layer is added.

Text prediction

In this paper we consider next character prediction on a Wikipedia text-corpus (A text corpus is a large and unstructured set of texts (nowadays usually electronically stored and processed) used to do statistical analysis and hypothesis testing) .

The total set is about 1.4 billion characters long, of which the final 10 million is used for testing. Each character is represented by one-out-of-N coding. We used 95 of the most common characters[2] (including small letters, capitals, numbers and punctuation), and one 'unknown' character, used to map any character not part of the 95 common ones, e.g. Cyrillic and Chinese characters. We need time in the order of 10 days to train a single network, largely due to the difficulty of exploiting massively parallel computing for SGD. Therefore we only tested three network instantiations[3]. Each experiment was run on a single GPU (NVIDIA GeForce GTX 680, 4GB RAM).

http://www.gabormelli.com/RKB/1-of-N_Coding_System

The task is as follows: given a sequence of text, predict the probability distribution of the next character. The used performance metric is the average number of bits-per-character (BPC), given by BPC = $-\log2(p_c)$ where p_c is the probability as predicted by the network of the correct next character.

https://stats.stackexchange.com/questions/211858/how-to-compute-bits-per-character-bpc

Network setups

The challenge in character-level language modelling lies in the great diversity and sheer number of words that are used.

All our networks have a number of neurons selected such that in total they each had approximately 4.9 million trainable parameters, which allowed us to make a comparison to other published work We considered three networks: a common RNN (2119 units), a 5-layer DRNN-1O (727 units per layer), and a 5-layer DRNN-AO (706 units per layer)[4]. Initial learning rates n_{0} were chosen at 0.5, except for the the top layer of the DRNN-1O, where we picked n_{0} = 0:25 (as we observed that the nodes started to saturate if we used a too high learning rate).

Results of the study:

Model	BPC test
RNN	1.610
DRNN-AO	1.557
DRNN-1O	1.541
MRNN	1.55
PAQ	1.51
Hutter Prize (current record) [12]	1.276
Human level (estimated) [18]	0.6 – 1.3

Table 1: Results on the Wikipedia character prediction task. The first three numbers are our measurements, the next two the results on the same dataset found in [19]. The bottom two numbers were not measured on the same text corpus.

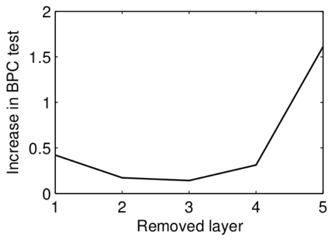


Figure 2: Increase in BPC on the test set from removing the output contribution of a single layer of the DRNN-AO.

From table 1, it becomes clear that common RNN performs worst, and DRNN-1O the best, with the DRNN-AO slightly worse.

To check how each layer influences performance in the case of the DRNN-AO, we performed tests in which the output of a single layer was set to zero. Fig 2 shows the corresponding results. Removing the bottom and top layers led to significant rise in BPC which shows that the contribution of the top layer is the most important, and that of the bottom layer second important. The intermediate layers contribute less to the direct output and seem to be more important in preprocessing the data for the top layer.

Text Generation with the model:

Using the probabilities output by the model, the study tried to form sentences by recursively sampling a new input character. The input phrase used was ‘The meaning of life is ‘ . This was done with three DRNN-AO networks, one intact, the second without the bottom hidden layer and the third without the top hidden layer. The predicted statements were as follows :

The meaning of life is the ”decorator of Rose”. The Ju along with its perspective character survive, which coincides with his eromine, water and colorful art called ”Charles VIII”.??In ”Inferno” (also 220: ”The second Note Game Magazine”, a comic at the Old Boys at the Earl of Jerusalem for two years) focused on expanded domestic differences from 60 mm Oregon launching, and are permitted to exchange guidance.	The meaning of life is impossible to unprecede ?Pok.{* PRER)!—KGOREMFHEAZ_CTX=R_M —S=6_5?&+——=7xp*=_5FJ4—13/TxI JX=—b28O=&4+E9F=&Z26__—R&N== Z8&A=58=84&T=RESTZINA=L&95Y_ 2O59&FP85=&&#=&H=S=Z_IO_=T _@—CBOM=6&9Y1=__9_5__	The meaning of life is man sistasteredsteris bus and nuster eril”n ton nis our ousNmachiselle here hereds?d toppstes impedred wisv.”-hor ens htls betwez rese, and Intantored wren in thoug and elit toren on the marcel, gos infand foldedsamps que help sasecre hon Roser and ens in respoted we frequen enctuivat herde pitched pitchugismissedre and loseflowered

Table 2: Three examples of text, generated by the DRNN-AO. The left one is generated by the intact network, the middle one by leaving out the contribution of the first layer, and the right one by leaving out the contribution of the top layer.

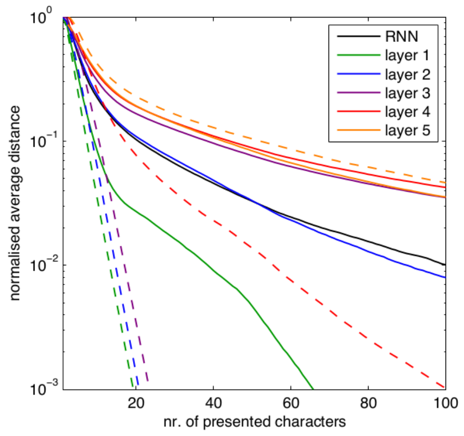
In the first column, the network was one with all hidden layers intact. Hence the model predicted fairly common words and formed sentences with correct short-term grammar, punctuations etc. Although not very sensible, the statements resemble natural language to a good extent.

In the second column, the network was one without the bottom hidden layer. The statements are mostly gibberish. The model becomes unstable and starts predicting rarely used letters and symbols. This demonstrates that the bottom layer captures the most basic text statistics of the dataset. On training it on a dataset of uncommon characters, the model starts predicting common letters and hence ‘normal text’, which confirms the aforementioned proposition about the bottom layer.

In the third column, the network lacks the top hidden layer. It predicts common characters and forms words of natural lengths which contain substrings of common syllables. However it seems to lack the capability of understanding text statistics of text longer than a word. Thus the top layers captures statistics of entire sentences.

Time Scale:

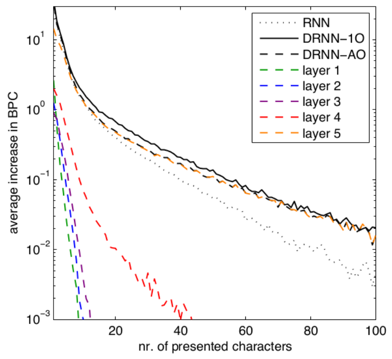
One of the main purposes of this paper is to introduce ,in RNNs, the ability to process data in several time scales. To find out at what time scale each layer operates, the following expt is conducted. Two networks are fed two strings as inputs, such that both the inputs are identical upto the first 100 characters after which a typo is introduced in one of them.
We then monitor the hidden states of each layer and the overall output to find out how long the typo-perturbation persists in the corresponding model i.e. how many characters does it take ,after the typo, for the model to forget the typo.
This is plotted by finding the euclidean distance between the states of the equivalent hidden layers of the perturbed and unperturbed model.



Here the dashed lines represent DRNN-AO and full lines DRNN-1O

DRNN-AO:
Since layers 1,2 and 3 need fewer characters than the other layers to forget the typo and essentially come back to a normal unperturbed state, we can say that the 1st 3 hidden layers have relatively shorter memory. Conversely the 4th and 5th layers need a lot more characters and hence have longer memory.
DRNN-1O
The top 3 layers here seem to have significantly longer memory which seems appropriate based on how the DRNN-1O network was trained.

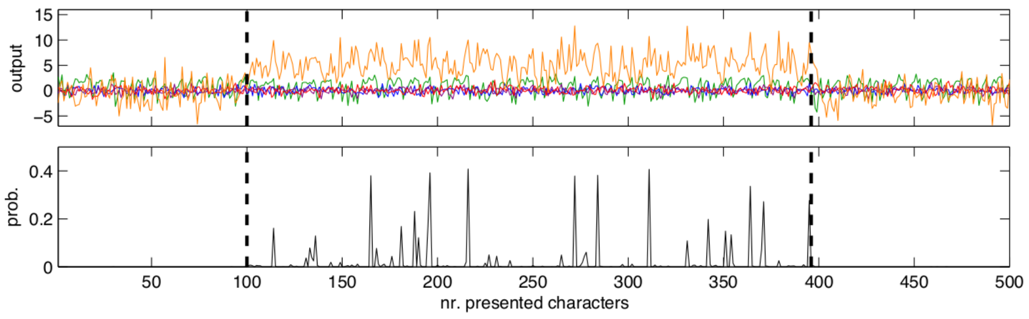
Effect of switching context:



The above figure depicts the effect on switching the context on the actual prediction accuracy. To do so we measured the average difference in BPC between normal text and a perturbed copy, in which we replaced the first 100 characters by text randomly sampled from elsewhere in the test set. This will give an indication of how long the lack of correct context lingers.
Here, DRNNs seem to recover more slowly from the context switch than the RNN, indicating that they employ a longer context for prediction. The time scales of the individual layers of the DRNN-AO are also depicted which largely confirms the result from the typo-perturbation test.

Testing long term memory:

Here the aim is to find out the ability of DRNNs to correctly predict the position of the closing parentheses, given a text sequence starting with the opening parentheses , even when the text sequence between them is quite long.



The 1st graph from the above figure plots the output traces for closing parentheses of the respective layers. The top most layer has pretty strong predictions for closing parentheses throughout the text sequence but they immediately drop once the sequence is closed.

Similarly the net output probability (lower graph in the figure) given by the model for closing parentheses shows momentary spikes throughout the sequence, but drops to almost zero everywhere else.
It is observed that DRNNs can remember opening brackets for text sequences longer than what they have been trained on, which means they don't just memorize the common length of text sequences between parenthesis, rather they model them as some pseudo stable attractor-like state.

Conclusion:

The paper successfully demonstrates that DRNNs are beneficial for character-level language modeling, reaching state-of-the-art performance for recurrent neural networks and boosting performance. The different layers in a DRNN provide a hierarchy of timescales. DRNNs have a memory of several hundreds of characters long and make it easier to to learn long-term relations between input characters than common RNNs.