Recurrent Neural Networks

Jose Gallego
University of Amsterdam
jose.gallegoposada@student.uva.nl

Abstract

In this work we explore the capabilities of Recurrent Neural Networks (RNNs) and Long Short-Term Networks (LSTMs) for language modelling. In the first part we apply this models to a synthetic dataset of palindrome numbers; and in the second part we use a stacked LSTM architecture to learn and generate text trained on real books. Our experiments show a clear advantage of LSTMs over RNNs for learning dependencies which span over very large time steps. This advantage in turn allows for a good performance in the text generation task.

1 Vanilla RNN versus LSTM

Before delving into the harder problem of applying LSTMs for Natural Language generation, we study the problem of predicting the last digit of a palindrome number. This means, for example, given the sequence `173037_`, we want our models to accurately predict `1`.

Question 1.1

In this section we use the following formulation for the RNN. Given a sequence of input vectors $\mathbf{x}^{(t)}$ for $t = 1, ..., T$, the network computes a sequence of hidden states $\mathbf{h}^{(t)}$ and a sequence of output vectors $\mathbf{p}^{(t)}$ using the following equations for timesteps $t = 1, ..., T$:

$$\begin{aligned} \mathbf{h}^{(t)} &= \tanh\left(\mathbf{W}_{hx}\mathbf{x}^{(t)} + \mathbf{W}_{hh}\mathbf{h}^{(t-1)} + \mathbf{b}_h\right) \\ \mathbf{p}^{(t)} &= \mathbf{W}_{oh}\mathbf{h}^{(t)} + \mathbf{b}_o \\ \hat{\mathbf{y}}^{(t)} &= \text{softmax}(\mathbf{p}^{(t)}) \end{aligned}$$

For the task of predicting the final palindrome number, we compute the standard cross-entropy loss only over the last timestep:

$$\mathcal{L} = -\sum_{k=1}^{K} \mathbf{y}_k^{(T)} \log \hat{\mathbf{y}}_k^{(T)} = -\log \hat{\mathbf{y}}_c^{(T)} = -\mathbf{p}_c^{(T)} + \log \sum_{l=1}^{K} e^{\mathbf{p}_l^{(T)}},$$

where k runs over the number of classes ($K = 10$ because we have ten digits) and c denotes the correct class at time T. In this expression, $\mathbf{y}^{(T)}$ denotes a one-hot vector of length K containing true labels.

Let us calculate the gradient of $\mathcal{L}$ wrt $\mathbf{W}_{oh}$. We δ_i^j represents the indicator whether $i = j$. $\mathbf{A}_{k\cdot}$ and $\mathbf{A}_{\cdot k}$ represent the k-th row and column of $\mathbf{A}$, respectively.

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_{oh}} = \sum_{k=1}^{K} \frac{\partial \mathcal{L}}{\partial \mathbf{p}_k^{(T)}} \frac{\partial \mathbf{p}_k^{(T)}}{\partial \mathbf{W}_{oh}}$$

$$\mathbf{p}_k^{(T)} = (\mathbf{W}_{oh})_{k\cdot}\, \mathbf{h}^{(T)} + (\mathbf{b}_o)_k$$

30th Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain.

$$\frac{\partial \mathbf{p}_k^{(T)}}{\partial \left(\mathbf{W}_{oh}\right)_{ij}} = \mathbf{h}_j^{(T)} \delta_i^k \quad \Rightarrow \quad \frac{\partial \mathbf{p}_k^{(T)}}{\partial \mathbf{W}_{oh}} = \begin{bmatrix} \mathbf{0} \\ \mathbf{h}_1^{(T)} \quad \cdots \quad \mathbf{h}_h^{(T)} \quad \cdots \quad \mathbf{h}_H^{(T)} \\ \mathbf{0} \end{bmatrix} \leftarrow k\text{-th row}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{p}_k^{(T)}} = -\delta_c^k + \hat{\mathbf{y}}_k^{(T)} \quad \Rightarrow \quad \frac{\partial \mathcal{L}}{\partial \mathbf{p}^{(T)}} = \hat{\mathbf{y}}^{(T)} - \mathbf{y}^{(T)}$$

Therefore, we can express the derivative concisely as:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_{oh}} = \left(\hat{\mathbf{y}}^{(T)} - \mathbf{y}^{(T)}\right) \mathbf{h}^{(T)\top}$$

Consider now the gradient of $\mathcal{L}$ wrt $\mathbf{W}_{hh}$:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_{hh}} = \sum_{l=1}^{H} \frac{\partial \mathcal{L}}{\partial \mathbf{h}_l^{(T)}} \frac{\partial \mathbf{h}_l^{(T)}}{\partial \mathbf{W}_{hh}} =: \sum_{l=1}^{H} \frac{\partial \mathcal{L}}{\partial \mathbf{h}_l^{(T)}} \, \mathbf{\Omega}_l^{(T)}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{h}_l^{(T)}} = \left(\mathbf{W}_{oh}{}^{\top}\right)_{l\cdot} \left(\hat{\mathbf{y}}^{(T)} - \mathbf{y}^{(T)}\right)$$

Let $\mathbf{S}_l^{(t)} = \left(\mathbf{W}_{hx}\right)_{l\cdot} \mathbf{x}^{(t)} + \left(\mathbf{W}_{hh}\right)_{l\cdot} \mathbf{h}^{(t-1)} + \left(\mathbf{b}_h\right)_l$.

$$\left(\mathbf{\Omega}_l^{(t)}\right)_{ij} = \frac{\partial \mathbf{h}_l^{(t)}}{\partial \left(\mathbf{W}_{hh}\right)_{ij}} = \frac{\partial \tanh\left(\mathbf{S}_l^{(t)}\right)}{\partial \left(\mathbf{W}_{hh}\right)_{ij}} = \frac{\partial \tanh\left(\mathbf{S}_l^{(t)}\right)}{\partial \mathbf{S}_l^{(t)}} \frac{\partial \mathbf{S}_l^{(t)}}{\partial \left(\mathbf{W}_{hh}\right)_{ij}} = \left(1 - \left(\mathbf{h}_l^{(t-1)}\right)^2\right) \frac{\partial \mathbf{S}_l^{(t)}}{\partial \left(\mathbf{W}_{hh}\right)_{ij}}$$

Now, the product rule gives us:

$$\begin{aligned}
\frac{\partial \mathbf{S}_l^{(t)}}{\partial \left(\mathbf{W}_{hh}\right)_{ij}} &= \frac{\partial \left(\mathbf{W}_{hh}\right)_{l\cdot} \mathbf{h}^{(t-1)}}{\partial \left(\mathbf{W}_{hh}\right)_{ij}} = \frac{\partial \sum_{\alpha=1}^{H} \left(\mathbf{W}_{hh}\right)_{l\alpha} \mathbf{h}_\alpha^{(t-1)}}{\partial \left(\mathbf{W}_{hh}\right)_{ij}} \\
&= \sum_{\alpha=1}^{H} \frac{\partial \left(\mathbf{W}_{hh}\right)_{l\alpha}}{\partial \left(\mathbf{W}_{hh}\right)_{ij}} \mathbf{h}_\alpha^{(t-1)} + \left(\mathbf{W}_{hh}\right)_{l\alpha} \frac{\partial \mathbf{h}_\alpha^{(t-1)}}{\partial \left(\mathbf{W}_{hh}\right)_{ij}} \\
&= \sum_{\alpha=1}^{H} \delta_i^l \delta_\alpha^j \mathbf{h}_\alpha^{(t-1)} + \sum_{\alpha=1}^{H} \left(\mathbf{W}_{hh}\right)_{l\alpha} \left(\mathbf{\Omega}_\alpha^{(t-1)}\right)_{ij} \\
&=: \left(\mathbf{\Theta}_l^{(t)}\right)_{ij} + \left(\mathbf{\Gamma}_l^{(t)}\right)_{ij}
\end{aligned}$$

Thus we can write:

$$\mathbf{\Omega}_l^{(t)} = \left(1 - \left(\mathbf{h}_l^{(t-1)}\right)^2\right) \left[\mathbf{\Theta}_l^{(t)} + \mathbf{\Gamma}_l^{(t)}\right]$$

Therefore,

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_{hh}} = \sum_{l=1}^{H} \left(\mathbf{W}_{oh}{}^{\top}\right)_{l\cdot} \left(\hat{\mathbf{y}}^{(T)} - \mathbf{y}^{(T)}\right) \mathbf{\Omega}_l^{(t)}$$

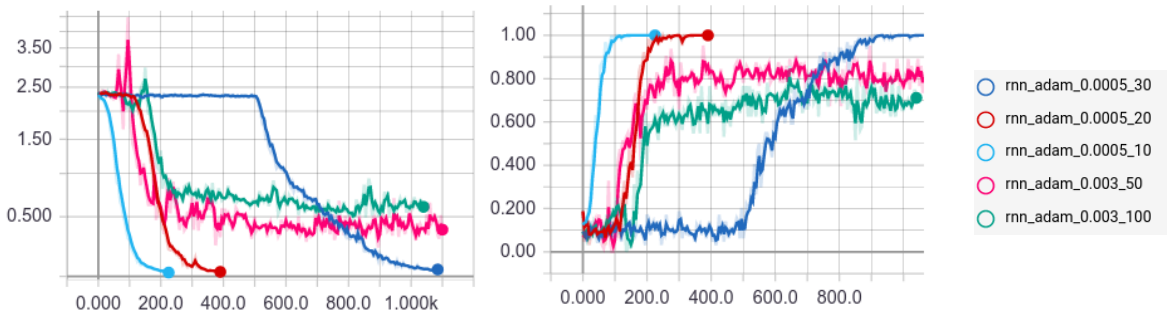
First, note that $\mathbf{\Omega}_l^{(t)}$ depends on $\mathbf{\Omega}_l^{(t-1)}$. Observe also that since $\mathbf{h}^{(0)}$ is initialized independent of $\mathbf{W}_{hh}$, the base case of the recursion $\mathbf{\Omega}_l^{(0)}$ vanishes. This recursion clearly displays the temporal dependency of the gradient of $\mathcal{L}$ wrt $\mathbf{W}_{hh}$ which, unlike $\mathbf{W}_{oh}$, has an impact on the loss over all time steps.

Consider now the eigendecomposition of $\mathbf{W}_{hh} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^{\top}$. It is straightforward to show that $\mathbf{W}_{hh}^t = \mathbf{U}\mathbf{\Lambda}^t\mathbf{U}^{\top}$. And thus eigenvalues which do not belong to $S^1(\mathbb{C})$ will either vanish or diverge. Since we are repeatedly multiplying $\mathbf{\Omega}_l^{(t)}$ by $\mathbf{W}_{hh}$ in the recursion shown above, the gradient $\nabla_{\mathbf{W}_{hh}} \mathcal{L}$ can vanish or explode as $t \to \infty$.

Question 1.2

In the supplementary code provided along with this document we implement the vanilla recurrent neural network as specified by the equations above in the file `vanilla_rnn.py`.

Question 1.3



(a) Loss (b) Accuracy

Figure 1: Statistics for the RNN models training.

As expected, we see that the RNN manages to learn fairly quickly dependencies which span over short time steps $T = 10$, $T = 20$. For $T = 30$, the optimization starts in a plateau but is eventually able to reach 100% accuracy. However, for models with dependencies over 50 or 100 time steps the performance degrades significantly.

Question 1.4

Gradient-based optimization comes in general with no guarantee for convergence to a global optimum with non-convex cost functions due to the presence of local optima and saddle points, at which the gradient vanishes and thus no parameter updates occur. Consider the optimization procedure as a ball travelling over the loss surface guided by noisy gradient information.

A first idea to improve convergence is to add *momentum* to the dynamics in a way such that we can damp oscillations in directions of high curvature by combining gradients with opposite signs; and at the same time build up speed in directions with consistent gradients. This can be interpreted as keeping track of the "velocity" of the ball at every instant and updating this velocity accordingly. This could, for instance, allow the optimization to escape a local optimum provided sufficient momentum had been built before entering the valley.

Another interesting feature of advanced optimization method is the use of *adaptive learning rates* per parameter. Compare the gradients of a neural network across different layers, generally, the gradients in the layers closer to the output have higher magnitudes than those closer to the input. This suggests that the effective learning rate for the parameter updates might change widely across different parameters. The idea of adaptive learning rates consists of scaling the global learning rate by a local gain which depends on whether the sign of the gradient for a given parameter has been changing constantly, in which case we would decrease the local gain; or whether the gradient for that parameter has a consistent sign and thus we allow the local gain to grow.

Question 1.5

(a) Imagine the LSTM unit as a set of pipes through which information can flow and each of the gates as a filter or amplifier for the incoming signal. Let us describe in more detail how this flow happens.

Forget gate This layer looks at the previous hidden state $\mathbf{h}^{(t-1)}$ and the current input $\mathbf{x}^{(t)}$ and decide what information to keep or discard from the cell state $\mathbf{c}^{(t-1)}$. Note that we use a sigmoid activation as we want to represent the *proportion* or percentage of information to retain from the cell state $\mathbf{f}^{(t)}$.

Input and Input Modulation gates This gates do a similar task as the previous one: decide what new information we will store in the cell state. First, a sigmoid layer determines which values will be updated (again use sigmoid to define a proportion of update to be applied) $\mathbf{i}^{(t)}$. Additionally, a tanh layer computes new candidates $\mathbf{g}^{(t)}$ that could be added to the cell state. The tanh activation pushes the values between -1 and 1 allowing for cell states to increase or decrease directly, while preserving the centering of the data and the scale of the output $\mathbf{g}$.

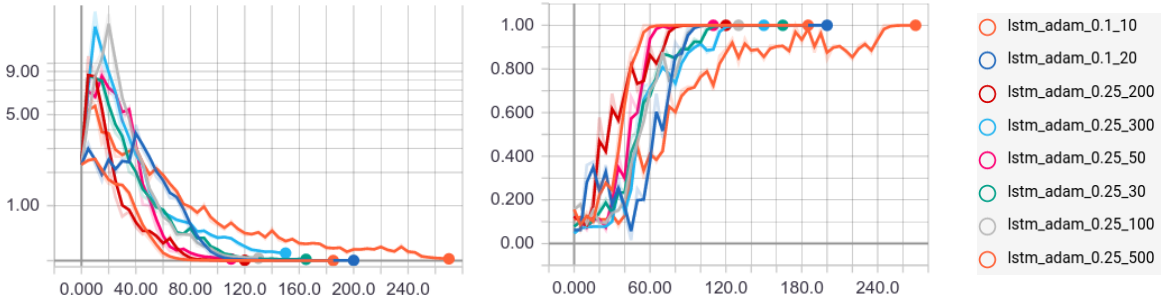
At this point we can update the cell state by forgetting from the old state $\mathbf{c}^{(t-1)}\mathbf{f}^{(t)}$ and add our new scaled candidates values $\mathbf{i}^{(t)}\mathbf{g}^{(t)}$.

Output gate Finally, the output $\mathbf{h}^{(t)}$ is computed as a product between a percentage factor $\mathbf{o}^{(t)}$ which decides which components we will output; and the tanh (to push the values to be between -1 and 1) activation of the cell state $\mathbf{c}^{(t)}$.

(b) Let d be the feature dimensionality, n the number of units in the LSTM, and k the number of output classes (which in the general case can be different from d). Each of $\{\mathbf{W}_{gx}, \mathbf{W}_{ix}, \mathbf{W}_{fx}, \mathbf{W}_{ox}\}$ has dn parameters. Each of $\{\mathbf{W}_{gh}, \mathbf{W}_{ih}, \mathbf{W}_{fh}, \mathbf{W}_{oh}\}$ has n^2 parameters. Each of $\{\mathbf{b}_g, \mathbf{b}_i, \mathbf{b}_f, \mathbf{b}_o\}$ has n parameters. $\mathbf{W}_{out}$ has size $k \times n$ and $\mathbf{b}_{out}$ is k-dimensional. Thus, the total number of trainable parameters is $4dn + 4n^2 + 4n + kn + k = 4n(d + n + 1) + k(n + 1)$.

Question 1.6

In the supplementary code provided along with this document we implement the LSTM as specified by the equations above in the file `lstm.py`.



(a) Loss (b) Accuracy

Figure 2: Statistics for the LSTM models training.

Figure 2 shows a clear advantage of LSTMs over RNNs for modelling long term dependencies. While RNNs were not able to achieve 100% accuracy for 30-time-step-long dependencies, the LSTMs are capable of solving the task perfectly for up to $T = 500$. This is not surprising, in the sense that the LSTM architecture is designed precisely aiming towards having more stable gradients (see Question 1.1) and thus having the ability to model longer dependencies better.

2 Recurrent Nets as Generative Model

Question 2.1

In this section we train a stacked LSTM architecture for a language modelling task. The goal of the network is to predict the next character in a sentence. The results shown below correspond to experiments with the RMSPROP optimizer, learning rate 0.0025, decay factor 0.96, an architecture of 2 stacked LSTMs with 128 units and a sequence length of 30. We train our model using the *Grimms' Fairy Tales* by the Grimm brothers, Jacob and Wilhelm. Additionally, we also train a model using *Don Quijote de la Mancha* by Miguel de Cervantes Saavedra, the most influential work of literature from the Spanish Golden Age and the entire Spanish literary canon. These two texts have widely different structure and linguistic complexity. We perform slight preprocessing of the original file to avoid the inclusion of spurious characters.

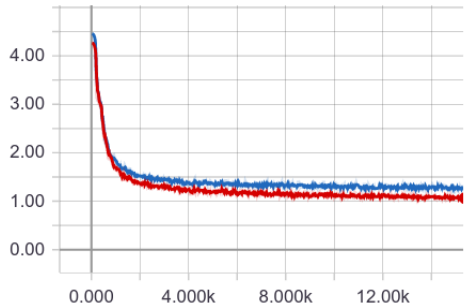


Figure 3: Training loss on *Grimms' Fairy Tales* (red) and *Don Quijote de la Mancha* (blue).

During training we sample our model by providing an initial random character and a hidden state initialized at zero. We implement two sampling mechanisms: taking the argmax of the logits or sampling from the categorical distribution generated thereby. For English text, the argmax method usually produces text like `the the the` or `and the and the` due to the high frequency of this patterns occurring in natural language. For this reason, the results displayed are sampled from the categorical distribution, where the logits were scaled by a temperature parameter to encourage more diversity. The bold characters before | are the randomly picked initializations.

Step	Sample
0	**7**|2qoU)qV:UUD7Pb9wg8HAJ:606u1(h4BMN3FuZ!(cu59;dQWSOT0-E0DientlAU0b6zK
3000	**d**| gone, and said, Wifhin, so no? I flew to me have I not have her shill came boy, the room
6000	**Q**|ueen, and they went as before, and restored and spread and beasts. And say needle, and had
9000	**Y**|ou sit into his eat and at an each of the king s bride should go into the golden bird am I!
12000	**J**|ust take home again, I may go into his husband in the courtyard of dress; and how she saw
15000	**G**|retel off again in the garden. With all my heart and more fellow, and they were once tired; b

Table 1: Samples during training on the *Grimms' Fairy Tales*.

Note how before training the sampled sequences are random samples from the vocabulary set, but as training progresses the model learns how the spacing and length of the words works. For instance, after 250 steps, the model is able to produce sequences as `fdrds w:hesyn uoge nsrpyodhenr diaeri adg,tsio`. At this point the placing of the commas and spaces is still problematic. However, this largely improves by step 750 when we start seeing some actual words and a correct use of punctuation: `the wall, chee he was with he fole:  I wall he ground`. Note also how over time the model is able to produce more complex words with correct spelling. An important remark is that the sentences are supposed to make syntactic sense locally (read small sequences of words) but will not necessarily be correct semantically or grammatically on a global scale. A way to improve on this could be to use a word embedding and train the model on sequences of words rather than just individual characters.

Step	Sample
0	**N**|0É0Aü,,As2PoGxLwí ÉcJéTy 3!Á-PeLK4A:ZyúHí?Q!449e'ñ¿hc'3Tc(ñrs;36üáhÑ'G
3000	**-**|dijo Sancho- que a decirlo de nadir como las comestamentes anday de lo que así se acompaña
6000	**ü**|era de no saber caballero de la ruega por su corta que la llama a poner las hermosas
9000	**Y**| padres y allí se consentiría de mi amigo. Pero, viendo que había sido salir de don Quijote
12000	**É**|stos, así de sus almas y abajos los dueñas de la buena caballería en la cual y pusiese a quien
15000	**L**|o que antes se puedo vencido bien hermo de escudero, la presencia con los diablos y responde

Table 2: Samples during training on *Don Quijote de la Mancha*.

Bonus

Another option for sampling from the model is to provide a initial sequence of characters, instead of just a single character, and use this sequence to initialize the cell state of the network, which can then be used to sample a possible continuation of the given sequence.

To be, or not to be, that is the question: Whether |he saw this little wild man be able to be quiet, it fare as she
History will be kind to me for I intend to |give it to the fire and the table of the stronger and wind; the sun was
Hansel and Gr|etel. Goodbye, Hans. Hans comes to the stable and a pity of this treatment, and slept over it,
Let T be a bounded linear operator in V, a vector space.| So she came to the road in his way to cook. At last he
Mas vale pajaro en mano que ver un ciento v|aluation, but in search and riberched to him. The salad on the fields

Table 3: Sentence completion after training on the *Grimms' Fairy Tales*.

We remark several aspects of these completions. First, note how a vast majority of the words are correctly spelled; and the places and punctuation signs are placed correctly. As expected, a very likely completion for the third sentence should start with completing the word Gretel. The fourth sentence shows how the model has learned how after a period a space and a capital letter should follow. Additionally, the model seems unaffected by the presence of samples with very different words (fourth example) and even different language (fifth example).

To be, or not to be, that is the question: Whether |lection electronic woreting yout Project Gutenberg in profiting
History will be kind to me for I intend to |o cHerbain no sola y qué nubviera que se le acompañaba verdad, im
Hansel and Gr|an Quijote, y la pendencia de todo lo que harán a don Quijote en dos molinas de sus ojos o de
Let T be a bounded linear operator in V, a vector space.| Grame doyaminables, camino y a todas las tercias de su
Mas vale pajaro en mano que ver un ciento v|liene, porque está aquí por esto, forzosamente al tercero en decir que

Table 4: Sentence completion after training on *Don Quijote de la Mancha*.

After training on *Don Quijote de la Mancha* a truly remarkable outcome appeared. Clearly, the last example in Spanish is completed in a sensible way. However, before the model was trained we ignored the fact that the dataset contained a copyright license regarding Project Gutenberg in English at the end of the file. Apparently, the model has learned that there is a sort of additional mode in the data distribution which corresponds to a different language, and for this reason completes the first example in a different way as an attempt to mimic the English patterns found in the initial part of this sequence.