Some fancy title: followed by some more text

Alyssa P. Hacker [1] Ben Bitdiddle [2] Lem E. Tweakit [2]

[1]Some Institute [2]Another Institute

Introduction

Recurrent neural networks (RNN) constitute a family of neural network architectures specialized to process sequential data. They do so by leveraging the simple idea of sharing parameters across the model[1]. They have been used in diverse domains for generating sequences such as music and text. They can be trained by processing real data sequences one step at a time and predicting what comes next. In practice, early RNNs designs are unable to store information about far past inputs. This fact diminishes their efficiency at modeling long structure. If the network's predictions are based only on the last few inputs, which are themselves predicted by the networks, then it has less chance to recover from past mistakes. The solution to this problem seems to be a better memory and especially a long-term memory for the network.

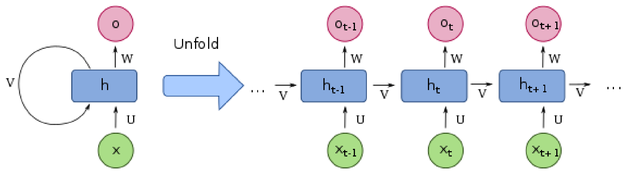
Long Short-Term Memory (LSTM) are RNN designed to solve this problem. They are better at storing informations than standard RNNs. It is important to note that LSTM gives state-of-the-art results in a variety of sequence processing tasks. This is the main reason why we decided to implement a LSTM for our text generation project. We worked on 4 differents dataset: Harry Potter's books, Lord of the ring's books, random quotes and a text from Shakespeare.

RNN

RNN comes from the following question: is there a neural network that depends on the full previous context that would model:

$$P(o_1, \ldots, o_T) = \prod_{t=1}^{T} P(o_t | o_1, \ldots o_{t-1})$$

They are feedforward neural networks with the addition of time dependency in the model by introducing edges that span the adjacent time steps in the network. At a given time, the nodes with recurrent edges receive input from the current data and from the output of the hidden layer in the previous state, see figure below. Thus, an input at time t can influence the output at time $t + \delta$.



$$h_t = \sigma(Ux_t + Vh_{(t-1)} + b_h),$$
$$o_t = \text{softmax}(Wh_t + b_o)$$

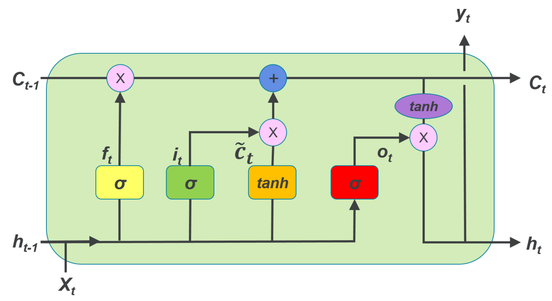
Here, U, V and W are weights matrix. The vectors b are bias parameters. Learning with RNN is challenging due to dependencies between long time steps. Consider the gradient with respect to h_t of $o_{t+\delta}$. How does it vary with δ? Following the graph above and applying the chain rule we can see that

$$\nabla_{h_t} o_{t+\delta} = \left(\prod_{k=t+1}^{t+\delta} V^T \text{diag}(1 - h_k^2) \right) \nabla_{h_{t+\delta}} o_{t+\delta}.$$

Thus, as δ grows, the gradient grows exponentially with V. If V is small or large than the gradient will either vanish or explode. This problem is well known. Solutions exist, which brings us to present the LSTM.

LSTM

The LSTM model has been introduced primarily to solve the vanishing and exploding gradients problem. This model is a RNN in which we replaced every hidden nodes by a *memory cell*.



Intuitively, RNN have *long-term memory* in the form of matrix weights, they change during the training encoding general knowledge about the data. RNN also have *short-term memory* in the form of activation passing from each node to successive ones. The memory cell introduced in the LSTM model provides storage for those memories. We now describe components of the cell following.[3]

- Gates (f_t, i_t, o_t): They are sigmoidal units that takes activation from the input x_t and the output of the hidden layer from previous state h_{t-1}. Note that f_t multiply the value of the previous cell c_{t-1}. The term *gate* stands for the literal meaning in the sense that if f_t is close to 0, then the gate is *closed* and the flow from the previous cell is cut off. If f_t is closed to 1 then all flow is passed through. The output to the hidden layer is $h_t = o_t \odot \tanh(c_t)$ where $\odot$ denote the pointwise multiplication.
- Cell state ($c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$): Cell state maintains information on the input. Also refered as the internal state, c_t has a self-connected edges with a fixed unit weight. This constant weight implies that the error can flow across time without vanishing or exploding.

Implementation Goal

For the implementation section of this project, we gathered text data that we found around the internet to train a sequence classifier and generate text sequences. The training of the classifier consist of identifying from which corpus between Harry Potter, Lord of the rings, some random quotes and Shakespeare, the sequence corresponds to. Further to this, we trained one model per sequence type for the text generation. Finally, we verified that our generated sequences were well classified by our classifier. The parameters we used for the models are shown in the table below.

Preprocessing

Before doing any sort or training, we had to do a bit of preprocessing on the data. First, we tokenized each corpus in sequences of 50 tokens. Then, we removed every capital letters to standardize the text. We built up a dictionary of every tokens ($\approx 60k$) in the datasets, this is our input space. Next, encode each token in a 256 dimensions vector with en embedding layer. We then feed the encoded vectors to our LSTMs with a hidden/cell state of 512 dimensions. The output dimension is 4 for the classifier and $\approx 60k$ for the text genetation.

Results

Classification of sequences

For the training of our classifier, we used the many-to-one architecture like the one below.

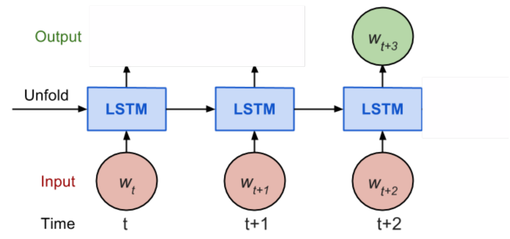


Figure: Many-to-one architecture.

We used the last output of the LSTM as our input for the classifier, disregarding all the other outputs. The last hidden state contains information about about all the sequence through the memory cell.