Mathematics Behind LSTM

1 Motivations

LSTM stands for Long-Short Term Memory. They are a type of Random Neural Network (RNN) and help avoid the vanishing/exploding gradient problem that RNNs commonly suffer from when they're unrolled.

In an LSTM we have a long-term memory and a short-term memory and both are utilised to help us make our predictions.

The long-term memory is unaffected by weights and biases and flows through with updates coming from the short-term memory. The lack of weights means that the long-term memory does not suffer with the vanishing/exploding problem.

The short-term memory is used to update the long-term memory and is the hidden state in the LSTM. The short-term memory is impacted by weights.

2 Weights

Weights are crucial in LSTMs, at each stage of the process, each value we use will have its own weight. These weights are calculated during the training process to minimise the loss function of the model which looks at the difference between the predicted and actual values.

3 Process

At each time step of the LSTM, it goes through 3 stages. We start with values for the long-term and short-term memories, these are initialised as 0 for the first time step as we haven't started yet and are updated throughout. These starting values for the long and short-term memories for the following time steps are determined by the output of the previous step. The input value is the corresponding value of the predictor variable for this time step.

3.1 Forget Gate

The first stage is called the 'Forget Gate'. We begin by doing,

$$x = (\text{STM Value * Its Weight}) + (\text{Input Value * Its Weight}) + \text{Bias}$$

This value of x is then put into the Sigmoid Activation Function. The Sigmoid Activation Function takes a value and returns a number between 0 and 1. The formula for the Sigmoid Activation Function is,

$$f(x) = \frac{e^x}{e^x+1}$$

We then multiply the value returned by the Sigmoid Activation Function by the value we have for the long-term memory to get an updated LTM.

This first stage determines how much of the long-term memory should be kept, hence the name 'Forget Gate'. If our value for x was very small, it's obvious that $f(x)$ would be very small and therefore the updated long-term memory would be close to 0 and all long-term memory forgotten. Similarly, if it was large then $f(x)$ would be close to 1 and all long-term memory would be kept.

3.2 Input Gate

The next step is called the 'Input Gate' and consists of two separate calculations. The first one combines the STM and Input values to create a potential long-term memory. We first calculate x in the same manner that we did earlier.

We now input this into the Tanh Activation function. The Tanh Activation Function takes a value and returns a number between -1 and 1. The formula for the Tanh Activation Function is,

$$f(x) = \frac{e^x-e^{-x}}{e^x+e^{-x}}$$

This is our value for the potential LTM.

We then need to determine how much of this new potential LTM is retained. Again, we caluclate x in the same way as before and then input this into the Sigmoid Activation Function. We'll call this percentage potential retained for referential purposes.
Lastly, we do,

$$\text{Additional LTM} = \text{Potential LTM * Percentage Potential Retained}$$

and add this onto the value we have for the LTM to get a New long-term memory.

3.3 Output Gate

In the last stage of the LSTM for this time step, we input our New LTM into the Tanh Activation Function. This is our potential STM.

We now need to determine how much of this potential STM is kept. This is done in the same manner as before, by calculating the percentage potential retained. We then do,

New LTM = Potential STM * Percentage Potential Retained

We now have our new values for the short-term and long-term memories that can be used for the next time step.

3.4 Next Steps

The above process is repeated for each time step, with the new input being the value of the predictor variable for this time step, and the new short-term and long-term memory values coming from the previous time step.

When we're then going to predict the value that we want at the end of the window, the value of the New short-term memory resulting from the Output Gate is our prediction.

4 How we're using the LSTM

We'll be looking at two LSTM models to predict pollution in Beijing, a univariate and a multivariate one. For the univariate model, it will be relatively simple and perform as above with the input value being the pollution of each hour. The multivariate model is a bit more complex as there are multiple features being inputted at each time step, in our case there will be eight. Rather than just learning about one feature and how this feature depends on previous values, it has to learn about how eight variables evolve and also about the relationship between these variables, adding another layer of complexity. However, in theory this should lead to an improved model as it can learn complex patterns within the data in order to make improved predictions. In the context of our problem this may prove to be very useful due to the relationships we'd expect between different features within the dataset, such as temperature and weather.

5 Why is this appropriate for our task

LSTMs are appropriate for the task at hand since there are many independent calculations going on within the model at each time step and these can be parallelised in order reduce the strain on any one processing unit and speed up a fairly complex model.

6 Remarks

At each time step we need to use the same weights for each stage within the time step.