LSTM Formulas - PyTorch Implementation

Tobias

January 29, 2025

1 Introduction

The Long Short-Term Memory (LSTM) is a type of recurrent neural network (RNN) designed to model sequences and dependencies over time. Below are the key equations that define the LSTM, following the PyTorch-style implementation.

2 LSTM Equations

2.1 Input Gate

The input gate decides which values to update in the cell state. In the PyTorch-style implementation, the input gate is computed as:

$$i_t = \sigma(W_{ii}x_t + b_{ii} + W_{hi}h_{t-1} + b_{hi}) \tag{1}$$

2.2 Forget Gate

The forget gate determines which parts of the previous cell state to keep:

$$f_t = \sigma(W_{if}x_t + b_{if} + W_{hf}h_{t-1} + b_{hf}) \tag{2}$$

2.3 Cell State Update

The candidate cell state is computed as:

$$\tilde{C}_t = \tanh(W_{ig}x_t + b_{ig} + W_{hg}h_{t-1} + b_{hg}) \tag{3}$$

The new cell state is updated using the forget and input gates:

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t \tag{4}$$

2.4 Output Gate

The output gate controls the hidden state and final output:

$$o_t = \sigma(W_{io}x_t + b_{io} + W_{ho}h_{t-1}) \tag{5}$$

The hidden state is then updated as:

$$h_t = o_t \odot \tanh(C_t) \tag{6}$$

3 Notation

- x_t: Input at time step t
- h_{t-1}: Hidden state from the previous time step
- C_t: Cell state at time step t
- $W_{ii}, W_{if}, W_{ig}, W_{io}$: Weight matrices for the input applied to input, forget, candidate cell, and output gates
- $W_{hi}, W_{hf}, W_{hg}, W_{ho}$: Weight matrices for the hidden state applied to input, forget, candidate cell, and output gates
- $b_{ii}, b_{if}, b_{ig}, b_{io}$: Biases for input, forget, candidate cell, and output gates
- σ: Sigmoid activation function
- tanh: Hyperbolic tangent activation function
- $\odot$: Element-wise multiplication (Hadamard product)

3.1 LSTM Inputs and Outputs

At each time step t, the Long Short-Term Memory (LSTM) unit takes the current input x_t and the previous hidden state h_{t-1} as inputs. It also utilizes the previous cell state C_{t-1} to compute two key outputs: the hidden state h_t and the cell state C_t. The hidden state h_t represents the output of the LSTM at time step t and is often used for tasks such as sequence prediction. The cell state C_t carries information across time steps, managing long-range dependencies. While both h_t and C_t are produced at each time step, the primary output used in further computations is typically the hidden state h_t.

3.2 Relevance propagation

3.2.1 Basics

First we'll take a step back and look at the basic building blocks of the LSTM and calculate the relevance propagation for each component. The relevance propagation is a technique used to understand the contribution of each input to the output of a neural network. By calculating the relevance propagation, we can identify which parts of the input are most relevant to the output, providing insights into the model's decision-making process.

4 Linear Layer with Tanh Activation

Consider a linear layer with 3 inputs and 1 output. The inputs and weights are defined as follows:

$$\begin{aligned} x_1 &= 0.1, \quad w_1 = 1 \\ x_2 &= 0.2, \quad w_2 = 2 \\ x_3 &= 0.3, \quad w_3 = 3 \end{aligned}$$

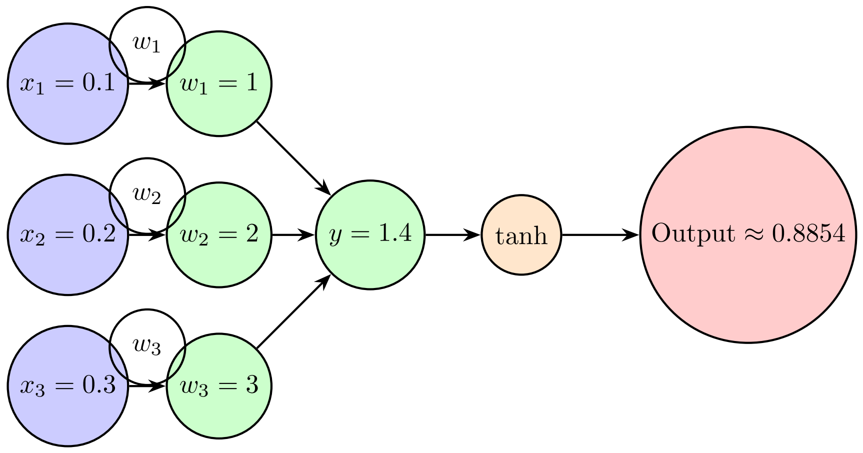
The output y of the linear layer can be calculated as:

$$y = w_1 x_1 + w_2 x_2 + w_3 x_3 = 1 \cdot 0.1 + 2 \cdot 0.2 + 3 \cdot 0.3 = 1.4$$

Next, we apply the hyperbolic tangent activation function:

$$\text{output} = \tanh(y) = \tanh(1.4) \approx 0.8854$$

5 Network Visualization



The layer-wise values for this are as follows:

$$\begin{aligned} R\left[x_1\right] &= \frac{\tanh(1.4)}{1.4} \cdot 1 \cdot 0.1 = 0.063 \\ R\left[x_2\right] &= \frac{\tanh(1.4)}{1.4} \cdot 2 \cdot 0.2 = 0.253 \\ R\left[x_3\right] &= \frac{\tanh(1.4)}{1.4} \cdot 3 \cdot 0.3 = 0.569 \end{aligned} \tag{7}$$

Till here, DeepLift and LRP are the same. However, the difference arises when there is a bias involved. Then in case of DeepLift the relevance is calculated as:

$$r[x_i] = \frac{\tanh(1.4) - \tanh(w_i * x_{i,\text{baseline}})}{1.4} w_i \cdot x_i \tag{8}$$

For LRP, without a baseline this results to:

$$r[x_i] = \tanh(1.4) \frac{x_i w_i}{1.4} \tag{9}$$

Or to generalize:

$$\mathbf{r}[\mathbf{x}] = \tanh(\mathbf{wx} + \mathbf{b}) \frac{\mathbf{w} \odot \mathbf{x}}{\mathbf{wx} + \mathbf{b}} \tag{10}$$

Unfortunately, this does violate the additivity principle (but this is how it's implemented in captum). We can conserve this though by not accounting for the bias in the linear layer.

$$\mathbf{r}[\mathbf{x}] = \tanh(\mathbf{wx} + \mathbf{b}) \frac{\mathbf{w} \odot \mathbf{x}}{\mathbf{wx}} \tag{11}$$

Code to follow along:

```
from captum.attr import LRP
from torch import nn
import torch
import torch.nn.functional as F
from shap import DeepExplainer

class SimpleModel(nn.Module):
    def __init__(self):
        super().__init__()
        self.tanh = nn.Tanh()
        self.fc = nn.Linear(in_features=3, out_features
            =1)

    def forward(self, x):
        x = self.fc(x)
        x = self.tanh(x)
        return x

model = SimpleModel()
weights = torch.tensor([[1., 2., 3.]], dtype=torch.
    float32)
bias = torch.tensor([0.0], dtype=torch.float32)
model.fc.weight.data = weights
model.fc.bias.data = bias

lrp = LRP(model)
```

```
    x = torch.tensor([[0.1, 0.2, 0.3]], dtype=torch.float32)
    attribution = lrp.attribute(x, target=0)
    output = model(x)

    # lrp attribution
    manual_attribution = torch.tanh(F.linear(weights, x,
        bias)) * weights * x / F.linear(weights, x, bias)

    # DeepLift attribution as used by shap
    x_baseline = torch.tensor([[0.0, 0.0, 0.0], [0.,0.,0.]],
         dtype=torch.float32)
    exp = DeepExplainer(model, x_baseline)
    values = exp(x)

    manual_values = (torch.tanh(F.linear(weights, x, bias))
        - torch.tanh(F.linear(weights, torch.Tensor([[0., 0.,
         0.]]), bias))) / F.linear(weights, x) * weights * x
```

For the DeepLift implementation, we can use the following:

$$\mathbf{r}[\mathbf{x}] = (\tanh(\mathbf{wx}+\mathbf{b}) - \tanh(\mathbf{wx}_{\text{baseline}}+\mathbf{b}))\frac{\mathbf{w}\odot\mathbf{x}}{\mathbf{wx}} \tag{12}$$

Now how does this translate to matrices $\mathbf{W}$? This works equally for matrices, but we need to account for the contribution of each neuron in the layer (in contrast to just one we had prior) attributing to the relevance scores to the ‘i‘-th output:

$$\mathbf{r}[\mathbf{x}]_i = \tanh(\mathbf{W}[i,:]\mathbf{x}+\mathbf{b})\frac{\mathbf{W}[i,:]\odot\mathbf{x}}{\mathbf{W}[i,:]\mathbf{x}} \tag{13}$$

5.0.1 LSTM

With this we are set to calculate the relevance propagation for the LSTM.

We work our way through the formulas (1), (todo: add others) and calculate the relevance propagation for each component. We start with

$$R\left[x_t\right] = \sigma(\mathbf{W}_{ii}x_t + \mathbf{b}_{ii} + \mathbf{W}_{hi}\mathbf{h}_{t-1})\frac{\mathbf{W}_{ii}\odot\mathbf{x}_t}{\mathbf{W}_{ii}\mathbf{x}_t + \mathbf{b}_{ii} + \mathbf{W}_{hi}\mathbf{h}_{t-1}} \tag{input gate - LRP}$$

$$\mathbf{r}\left[\mathbf{x}_t\right] = \sigma(\mathbf{W}_{ii}x_t + \mathbf{b}_{ii} + \mathbf{W}_{hi}\mathbf{h}_{t-1})\frac{\mathbf{W}_{ii}\odot\mathbf{x}_t}{\mathbf{W}_{ii}\mathbf{x}_t} \tag{input gate - Adjusted LRP}$$

shap always calculates both: $\mathbf{r}\left[\mathbf{x}_t\right]$ and $\mathbf{r}\left[\mathbf{h}_{t-1}\right]$. input gate - DeepLift - Rx

$$\begin{aligned}\mathbf{r}\left[\mathbf{x}_t\right] = (\sigma(\mathbf{W}_{ii}\mathbf{x}_t + \mathbf{b}_{ii} + \mathbf{W}_{hi}\mathbf{h}_{t-1}) - \\ \sigma(\mathbf{W}_{ii}\mathbf{x}_{\text{baseline}} + \mathbf{b}_{ii} + \mathbf{W}_{hi}\mathbf{h}_{\text{baseline}} + \mathbf{b}_{hi})) \\ \frac{\mathbf{W}_{ii}\odot\mathbf{x}_t}{\mathbf{W}_{ii}\mathbf{x}_t + \mathbf{W}_{hi}\mathbf{h}_{t-1}}\end{aligned} \tag{14}$$

input gate - DeepLift - Rh

$$\begin{aligned}\mathbf{r}\left[\mathbf{h}_{t-1}\right] = (\sigma(&\mathbf{W}_{ii}\mathbf{x}_t + \mathbf{b}_{ii} + \mathbf{W}_{hi}\mathbf{h}_{t-1} + \mathbf{b}_{hi}) - \\ &\sigma(\mathbf{W}_{ii}\mathbf{x}_{\text{base}} + \mathbf{b}_{ii} + \mathbf{W}_{hi}\mathbf{h}_{\text{base}} + \mathbf{b}_{hi}))\cdot \\ &\frac{\mathbf{W}_{hi} \odot \mathbf{h}_{t-1}}{\mathbf{W}_{ii}\mathbf{x}_t + \mathbf{W}_{hi}\mathbf{h}_{t-1}}\end{aligned} \tag{15}$$

Forget Gate (works analogously to Input gate):
forget gate - DeepLift - Rx

$$\begin{aligned}\mathbf{r}\left[\mathbf{x}_t\right] = (\sigma(&\mathbf{W}_{if}\mathbf{x}_t + \mathbf{b}_{if} + \mathbf{W}_{hi}\mathbf{h}_{t-1}) - \\ &\sigma(\mathbf{W}_{if}\mathbf{x}_{\text{baseline}} + \mathbf{b}_{if} + \mathbf{W}_{hf}\mathbf{h}_{\text{baseline}} + \mathbf{b}_{hf})) \\ &\frac{\mathbf{W}_{if} \odot \mathbf{x}_t}{\mathbf{W}_{if}\mathbf{x}_t + \mathbf{W}_{hf}\mathbf{h}_{t-1}}\end{aligned} \tag{16}$$

forget gate - DeepLift - Rh

$$\begin{aligned}\mathbf{r}\left[\mathbf{h}_{t-1}\right] = (\sigma(&\mathbf{W}_{ii}\mathbf{x}_t + \mathbf{b}_{ii} + \mathbf{W}_{hi}\mathbf{h}_{t-1} + \mathbf{b}_{hi}) - \\ &\sigma(\mathbf{W}_{ii}\mathbf{x}_{\text{base}} + \mathbf{b}_{ii} + \mathbf{W}_{hi}\mathbf{h}_{\text{base}} + \mathbf{b}_{hi}))\cdot \\ &\frac{\mathbf{W}_{hi} \odot \mathbf{h}_{t-1}}{\mathbf{W}_{ii}\mathbf{x}_t + \mathbf{W}_{hi}\mathbf{h}_{t-1}}\end{aligned} \tag{17}$$

Before we continue with the cell status update, we need to revise how shapley values are calculated:

$$\phi_i = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!(|N| - |S| - 1)!}{|N|!}(f(S \cup \{i\}) - f(S)) \tag{18}$$

Let's say we have a multiplication which is a function with two parameters:

$$f(x, y) = x \cdot y \tag{19}$$

then we can calculate the shapley values for f as:

$$\phi_x = \frac{1}{2}(f(x, y) - f(b_x, y)) + \frac{1}{2}(f(x, b_y) - f(b_x, b_y)) \tag{20}$$

where b_x and b_y are the baseline values for x and y respectively. The prefactor $\frac{1}{2}$ is due to the fact that we have two parameters and $|S| = 1$. Analogously

$$\phi_y = \frac{1}{2}(f(x, y) - f(x, b_y)) + \frac{1}{2}(f(b_x, y) - f(b_x, b_y)) \tag{21}$$

If we apply this to the SECOND PART OF the cell state update, we get:

$$\mathbf{R}[\mathbf{i}_t] = \frac{1}{2}\left[\mathbf{i}_t \odot \tilde{\mathbf{C}}_t - \mathbf{i}_{t,b} \odot \tilde{\mathbf{C}}_t + \mathbf{i}_t \odot \tilde{\mathbf{C}}_{t,b} - \mathbf{i}_{t,b} \odot \tilde{\mathbf{C}}_{t,b}\right] \tag{22}$$

In order calculate the relevance propagation we'll break it down here into the different components. The relevance propagation of equation (1) is calculated as: