5.2 Back Propagation Through Time(BPTT)

Nodes of our computational graph include the parameters U, V, W, b, c as well as the sequence of nodes indexed by t for $x^{(t)}$, $h^{(t)}$, $o^{(t)}$ and $L^{(t)}$.

For each node N we need to compute the gradient $\nabla_N L$ recursively.

Start recursion with nodes immediately preceding final loss

We assume that the outputs $o^{(t)}$ are used as arguments to softmax to obtain vector y^ of probabilities over the output.

Also that loss is the negative log-likelihood of the true target $y^{(t)}$ given the input so far.

The gradient on the outputs at time step t is, for all i, t is as follows:

$$\boxed{(\nabla_{\boldsymbol{o}^{(t)}} L)_i = \frac{\partial L}{\partial o_i^{(t)}} = \frac{\partial L}{\partial L^{(t)}} \frac{\partial L^{(t)}}{\partial o_i^{(t)}} = \hat{y}_i^{(t)} - \mathbf{1}_{i,y^{(t)}}}$$

Work backwards starting from the end of the sequence. At the final time step τ, h(τ) has only o(τ) as a descendent.

$$\boxed{\nabla_{\boldsymbol{h}^{(\tau)}} L = \boldsymbol{V}^\top \nabla_{\boldsymbol{o}^{(\tau)}} L}$$

In other words, the output at the final time step τ depends only on the final hidden state, and not on any future hidden states or inputs.

Iterate backwards in time iterating to backpropagate gradients through time from t =τ-1 down to t =1

$$\begin{aligned}\nabla_{\boldsymbol{h}^{(t)}} L &= \left(\frac{\partial \boldsymbol{h}^{(t+1)}}{\partial \boldsymbol{h}^{(t)}}\right)^\top (\nabla_{\boldsymbol{h}^{(t+1)}} L) + \left(\frac{\partial \boldsymbol{o}^{(t)}}{\partial \boldsymbol{h}^{(t)}}\right)^\top (\nabla_{\boldsymbol{o}^{(t)}} L) \\ &= \boldsymbol{W}^\top (\nabla_{\boldsymbol{h}^{(t+1)}} L) \operatorname{diag}\left(1 - \left(\boldsymbol{h}^{(t+1)}\right)^2\right) + \boldsymbol{V}^\top (\nabla_{\boldsymbol{o}^{(t)}} L)\end{aligned}$$

where $\operatorname{diag}\left(1 - (\boldsymbol{h}^{(t+1)})^2\right)$ indicates a diagonal matrix with elements within parentheses. This is the Jacobian of the hyperbolic tangent associated with the hidden unit i at time $t+1$

Once gradients on internal nodes of the computational graph are obtained, we can obtain gradients on the parameters.

$$
\begin{aligned}
\nabla_{\boldsymbol{c}} L &= \sum_t \left(\frac{\partial \boldsymbol{o}^{(t)}}{\partial \boldsymbol{c}}\right)^{\top} \nabla_{\boldsymbol{o}^{(t)}} L = \sum_t \nabla_{\boldsymbol{o}^{(t)}} L \\
\nabla_{\boldsymbol{b}} L &= \sum_t \left(\frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{b}^{(t)}}\right)^{\top} \nabla_{\boldsymbol{h}^{(t)}} L = \sum_t \operatorname{diag}\left(1 - \left(\boldsymbol{h}^{(t)}\right)^2\right) \nabla_{\boldsymbol{h}^{(t)}} L \\
\nabla_{\boldsymbol{V}} L &= \sum_t \sum_i \left(\frac{\partial L}{\partial o_i^{(t)}}\right) \nabla_{\boldsymbol{V}} o_i^{(t)} = \sum_t \left(\nabla_{\boldsymbol{o}^{(t)}} L\right) \boldsymbol{h}^{(t)\top} \\
\nabla_{\boldsymbol{W}} L &= \sum_t \sum_i \left(\frac{\partial L}{\partial h_i^{(t)}}\right) \nabla_{\boldsymbol{W}^{(t)}} h_i^{(t)} \\
&= \sum_t \operatorname{diag}\left(1 - \left(\boldsymbol{h}^{(t)}\right)^2\right) \left(\nabla_{\boldsymbol{h}^{(t)}} L\right) \boldsymbol{h}^{(t-1)\top} \\
\nabla_{\boldsymbol{U}} L &= \sum_t \sum_i \left(\frac{\partial L}{\partial h_i^{(t)}}\right) \nabla_{\boldsymbol{U}^{(t)}} h_i^{(t)} \\
&= \sum_t \operatorname{diag}\left(1 - \left(\boldsymbol{h}^{(t)}\right)^2\right) \left(\nabla_{\boldsymbol{h}^{(t)}} L\right) \boldsymbol{x}^{(t)\top}
\end{aligned}
$$

Vanishing/Exploding Gradients Problem:

1. **Understanding the Problems**

Vanishing –

As the backpropagation algorithm advances downwards(or backward) from the output layer towards the input layer, the gradients often get smaller and smaller and approach zero which eventually leaves the weights of the initial or lower layers nearly unchanged. As a result, the gradient descent never converges to the optimum. This is known as the vanishing gradients problem.

Exploding –

On the contrary, in some cases, the gradients keep on getting larger and larger as the backpropagation algorithm progresses. This, in turn, causes very large weight updates and causes the gradient descent to diverge. This is known as the exploding gradients problem.

Exploding	Vanishing
• There is an exponential growth in the model parameters. • The model weights may become NaN during training. • The model experiences avalanche learning.(the model's accuracy "avalanches" or rapidly declines when presented with new and unfamiliar data.)	• The parameters of the higher layers change significantly whereas the parameters of lower layers would not change much (or not at all). • The model weights may become 0 during training. • The model learns very slowly and perhaps the training stagnates at a very early stage just after a few iterations.

Methods that are proposed to overcome the vanishing gradient problem are:

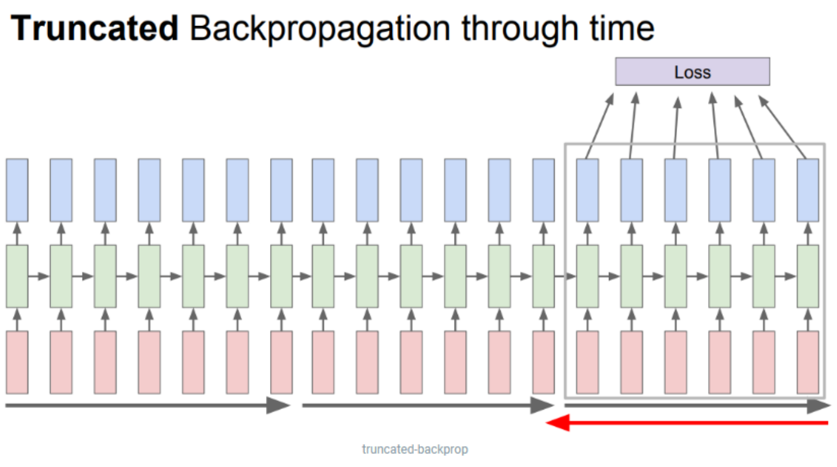
- Residual neural networks (ResNets)
- Multi-level hierarchy
- Long short term memory (LSTM)
- Faster hardware
- ReLU
- Batch normalization

Methods that are proposed to overcome the exploding gradient problem are:

- Use LSTM network
- Use Gradient clipping
- Use Regularization
- Redesign the neural network

Truncated Backpropagation Through Time

Truncated Backpropagation Through Time (truncated BPTT) is a widespread method for learning recurrent computational graphs. Truncated BPTT keeps the computational benefits of Backpropagation Through Time (BPTT) while relieving the need for a complete backtrack through the whole data sequence at every step.



Reduced memory requirements: By truncating the backpropagation algorithm, TBPTT reduces the amount of memory required to store intermediate activations during training. This is particularly important for RNNs, which can have very long sequences of inputs.

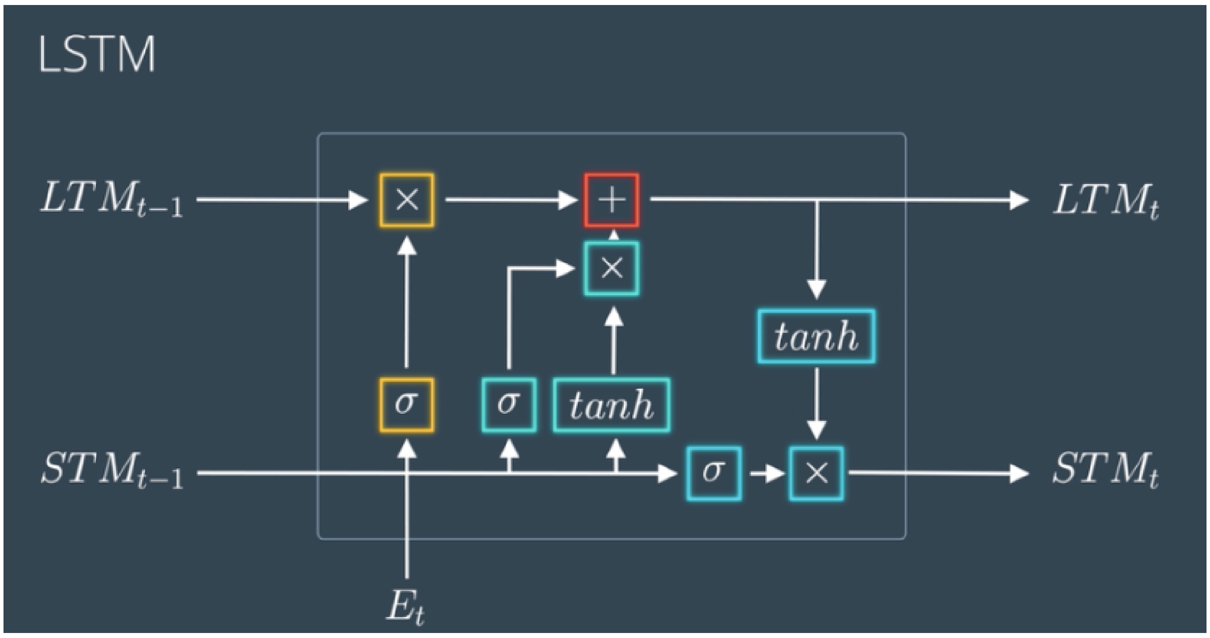
Faster training: By reducing the number of time steps over which the backpropagation algorithm is applied, TBPTT can speed up the training process. This can be especially useful for large and complex models.

Improved generalization: By truncating the backpropagation algorithm, TBPTT can help prevent overfitting by limiting the amount of influence that earlier time steps have on the final output. This can help the model generalize better to new data.

Flexibility: The number of time steps used in TBPTT can be adjusted to balance between computational efficiency and model performance. This allows for greater flexibility in designing and training RNNs.

In the backpropagation training, there is a forward pass and a backward pass through the entire sequence to compute the loss and the gradient. By taking a window, we also improve the training performance from the training duration aspect.

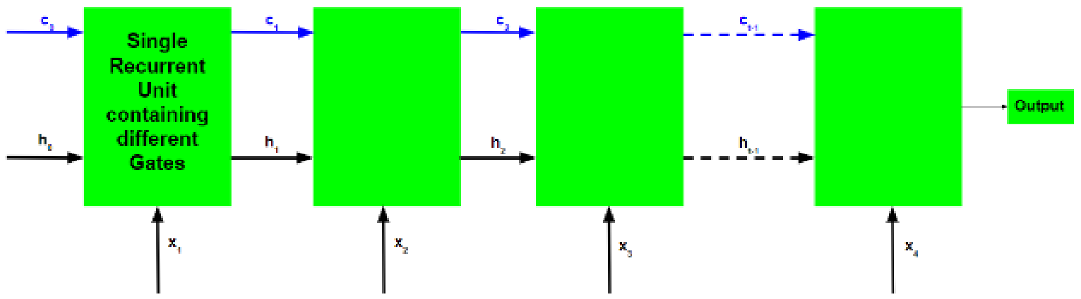
<u>The architecture of LSTM:</u>
LSTMs deal with both Long Term Memory (LTM) and Short Term Memory (STM) and for making the calculations simple and effective it uses the concept of gates.



LSTM networks are the most commonly used variation of Recurrent Neural Networks (RNNs). The critical component of the LSTM is the memory cell and the gates (including the forget gate but also the input gate), inner contents of the memory cell are modulated by the input gates and forget gates. Assuming that both of the segue he are closed, the contents of the memory cell will remain unmodified between one time-step and the next gradients gating structure allows information to be retained across many time-steps, and consequently also allows group that to flow across many time-steps. This allows the LSTM model to overcome the vanishing gradient properly occurs with most Recurrent Neural Network models.

Forget Gate(f): At forget gate the input is combined with the previous output to generate a fraction between 0 and 1, that determines how much of the previous state need to be preserved (or in other words, how much of the state should be forgotten). This output is then multiplied with the previous state. Note: An activation output of 1.0 means “remember everything” and activation output of 0.0 means “forget everything.” From a different perspective, a better name for the forget gate might be the “remember gate”
Input Gate(i): Input gate operates on the same signals as the forget gate, but here the objective is to decide which new information is going to enter the state of LSTM. The output of the input gate (again a fraction between 0 and 1) is multiplied with the output of tan h block that produces the new values that must be added to previous state. This gated vector is then added to previous state to generate current state

Input Modulation Gate(g): It is often considered as a sub-part of the input gate and much literature on LSTM's does not even mention it and assume it is inside the Input gate. It is used to modulate the information that the Input gate will write onto the Internal State Cell by adding non-linearity to the information and making the information Zero-mean. This is done to reduce the learning time as Zero-mean input has faster convergence. Although this gate's actions are less important than the others and are often treated as a finesse-providing concept, it is good practice to include this gate in the structure of the LSTM unit.
Output Gate(o): At output gate, the input and previous state are gated as before to generate another scaling fraction that is combined with the output of tanh block that brings the current state. This output is then given out. The output and state are fed back into the LSTM block.



In LSTM's we can selectively read, write and forget information by regulating the flow of information using gates.

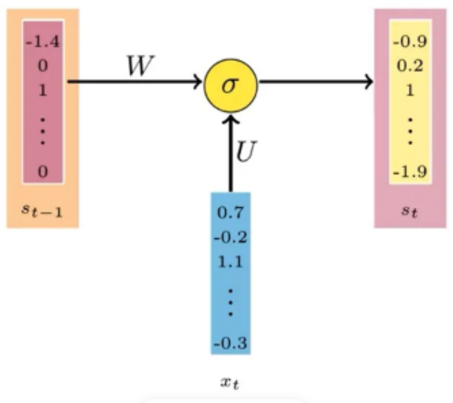
In the following few sections, we will discuss how we can implement the selective read, write and forget strategy. We will also discuss how do we know which information to read and which information to forget.

Selective Write

In the vanilla RNN version, the hidden representation ($\mathbf{s_t}$) computed as a function of the output of the previous time step hidden representation ($\mathbf{s_{t-1}}$) and current input ($\mathbf{x_t}$) along with bias ($\mathbf{b}$).

$$s_t = \sigma(Ux_t + W\mathbf{s_{t-1}} + b)$$

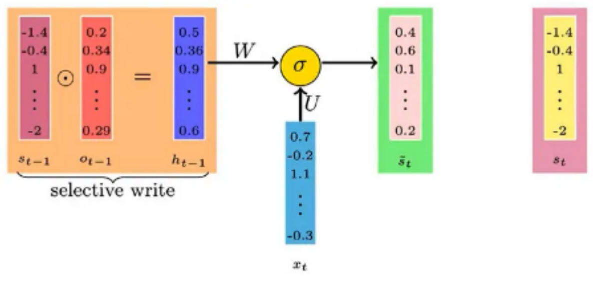
Here, we are taking all the values of $\mathbf{s_{t-1}}$ and computing the hidden state representation at the current time ($\mathbf{s_t}$).



In Selective Write, instead of writing all the information in s_{t-1} to compute the hidden representation (s_t). we could pass only some information about s_{t-1} to the next state to compute s_t.

Selective Read

After computing the new vector $\mathbf{h_{t-1}}$ we will compute an intermediatory hidden state vector $\mathbf{\check{S}_t}$ (marked in green). In this section, we will discuss how to implement selective read to get our final hidden state $\mathbf{s_t}$.



The mathematical equation for $\mathbf{\check{S}_t}$ is given below:

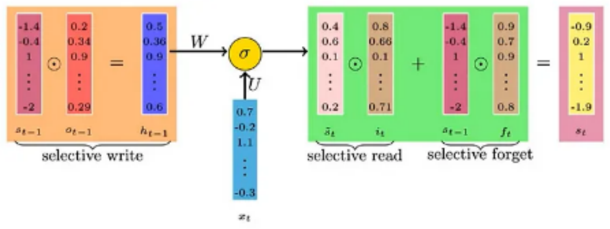
$$\tilde{s}_t = \sigma(Ux_t + Wh_{t-1} + b)$$

Št captures all the information from the previous state h_{t-1} and the current input x_t. However, we may not want to use all the new information and only selectively read from it before constructing the new cell structure. i.e… we would like to read only some information from Št to compute the s_t.

Selective Forget

In this section, we will discuss how we will compute the current state vector $\mathbf{s_t}$ by combining $\mathbf{s_{t-1}}$ and $\mathbf{\check{S}_t}$.

The **Forget Gate** $\mathbf{f_t}$ decides what fraction of information to be retained or discarded from $\mathbf{s_{t-1}}$ hidden vector.



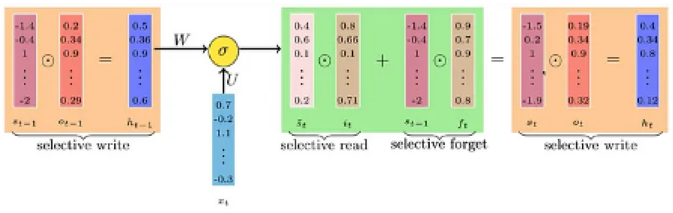
Selective Forget

The mathematical equation for forget gate $\mathbf{f_t}$ is given below:

$$f_t = \sigma(U_f x_t + W_f h_{t-1} + b_f)$$

In forget gate, we pass the previous time step hidden state information h_{t-1} and the current input x_t along with a bias into a sigmoid function. The output of the computation will between 0–1 and it will decide what information to be retained or discarded. If the value is closer to 0 means to discard and if it’s closer to 1 means to retain.

The final illustration would look like this:



The full set of equations looks like this:

Gates:

$$o_t = \sigma(W_o h_{t-1} + U_o x_t + b_o)$$
$$i_t = \sigma(W_i h_{t-1} + U_i x_t + b_i)$$
$$f_t = \sigma(W_f h_{t-1} + U_f x_t + b_f)$$

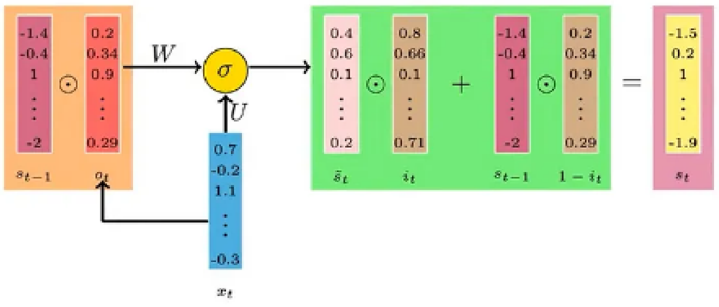
States:

$$\tilde{s}_t = \sigma(W h_{t-1} + U x_t + b)$$
$$s_t = f_t \odot s_{t-1} + i_t \odot \tilde{s}_t$$
$$h_t = o_t \odot \sigma(s_t)$$

Gated Recurrent Units — GRU's

In this section, we will briefly discuss the intuition behind GRU. Gated Recurrent Units is another popular variant of LSTM. GRU uses fewer gates.

In Gated Recurrent Units just like LSTM, we have an output gate $\mathbf{o_{t-1}}$ controlling what information is going to the next hidden state. Similarly, we also have an input gate $\mathbf{i_t}$ controlling what information flows in from the current input.


W
σ
U
s_{t-1}
o_t
x_t
$\tilde{s}_t$
i_t
s_{t-1}
$1 - i_t$
s_t